ADDALY · THE AI CLASSROOM

Knowing If It Works

The least glamorous skill in applied AI, and the one that decides whether your thing actually works.

---

8 modules · 76 lessons · 28 figures · 8 case studies · 57,695 words · about 11 hours

Dr Mustafa Mun
Author and editor

ABOUT THIS BOOK

Knowing If It Works, 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/evaluating-ai. 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 counts as evidence

Turn a vague claim that a feature works into a fixed, labelled, held-out set with a stated agreement figure and a named baseline, and say what size of difference a set that size can and cannot detect

1. A demo is not a measurement
2. The decision the number has to serve
3. A hundred examples, made by hand
4. Where the examples come from
5. Writing a labelling guide two people can agree on
6. How much your labellers actually agree
7. Test cases you generate, and what they cannot tell you
8. Two sets, and the ways they get contaminated
9. The number above it and the number below it
10. How wrong your number probably is
11. Case study: Fifteen questions and a batch starting Monday
12. Module test

2. Metrics that mean something

Choose a metric that matches the decision at hand, set a threshold from a real confusion matrix or a precision-recall curve, say whether a model's 0.8 means anything about probability, and separate a retrieval failure from a generation failure in a system that does both

1. What to measure when the output is prose
2. Precision and recall, worked on real numbers
3. When an item has several right answers
4. What a curve says, and what its one number leaves out
5. Moving the threshold is a business decision
6. Making a 0.8 mean eighty per cent
7. When the output is a quantity
8. Scoring a ranked list
9. Three questions, not one, for a retrieval system
10. Public benchmarks, and what they are for
11. Case study: The nurse who can read 250 messages a day
12. Module test

3. Deciding whether a difference is real

Decide whether an observed difference between two systems is larger than the noise in your set, compute an interval for any metric you can calculate using the bootstrap, and state in advance how many items the comparison needs to detect the effect you would act on

1. Asking what pure chance would have produced
2. What a p-value is, and the four things it is not
3. Sizing the set before you run it
4. The bootstrap, and the three places it lies
5. Five hundred rows, forty users
6. Twenty slices and one guaranteed surprise
7. Watching the dashboard until it says yes
8. Getting more signal from the items you already have
9. Better in every group and worse overall
10. Case study: Eighty-seven beats eighty-two, or does it
11. Module test

4. The systems you will actually be handed

Design a defensible evaluation for a summariser, a translator, a code generator, a structured extractor, a multi-turn assistant, a ranking system, a vision or speech model, a document pipeline or a tabular model, and name the specific failure mode each design exists to catch

1. Summaries: what was invented, and what was left out
2. Translation, and the languages nobody measures
3. Code: run it, in a box, against tests it has not seen
4. Structured extraction, where normalisation is most of the work
5. Multi-turn: score the session, not the turn
6. Ranking systems, where the labels are clicks and the clicks are biased
7. Images: the background the model is actually reading
8. Speech: word error rate, and the words that matter
9. Documents: stages that multiply, and checks that need no labels
10. Tables, cross-validation, and the leak that gives you 0.99
11. Case study: Ninety-eight per cent of the fields, sixty-one per cent of the forms
12. Module test

5. Automating judgement

Build a layered automatic check with cheap deterministic assertions underneath a rubric-driven model judge, state the agreement figure that licenses trusting the judge's numbers, and control the position bias, self-preference and cost that judging brings with it

1. LLM-as-judge, and where it misleads you
2. Turning a quality you can feel into rules a machine can apply
3. Gold answers, and asking whether two answers agree
4. Proving the judge agrees with you, and re-proving it
5. Turning pairwise wins into a ranking you can defend
6. The checks that cost nothing and catch most of it
7. Checking an answer against the passages it was given
8. What judging costs, and how to pay a tenth of it
9. Testing what happens when somebody tries
10. When the output is a sequence of actions
11. Case study: The judge that gave everything an eight
12. Module test

6. Evaluation in the working loop

Run evaluation as part of development — a harness in CI that survives a stochastic model, logging complete enough to diagnose a failure you did not reproduce, a structured error-analysis session that names causes rather than symptoms, and a staged release with a metric that can stop it

1. The harness, and why you should build the small one first
2. An evaluation that runs on every change without going red at random
3. Recording enough to diagnose a failure you did not see happen
4. Regression testing a prompt
5. Reading fifty failures
6. A/B testing a feature honestly
7. Getting evidence from live traffic without exposing users
8. The signals users give you without being asked
9. Sampling live traffic, and keeping reviewers honest
10. Case study: The gate that went red on a README
11. Module test

7. Evidence other people can act on

Assemble an evaluation record a stranger can audit — documented set, per-group results, stated limitations, reproducible run — interrogate a supplier's claims with a set of your own, and say clearly where a quality judgement is contested rather than merely unmeasured

1. The document that travels with the system
2. Buying a model you will depend on
3. Why the builder's own numbers are weak evidence
4. What the emerging rules ask for, and what they do not settle
5. Measuring the worst case, not the average one
6. Measuring the human in the loop
7. Testing whether the system gives away what it was given
8. When the model is right and the product still fails
9. When people disagree about what good means
10. Case study: The card said ninety-six per cent
11. Module test

8. Living with it

Keep a shipped system honest over time — monitor for drift, measure the tail of cost and latency rather than the mean, measure quality separately for each group of users, convert incidents into permanent test items, and report a result with its uncertainty stated

1. Cost, latency, and the model that changed under you
2. The tail is what your users feel
3. Watching a system you can no longer test
4. The system that trains on its own output
5. Refreshing a set without breaking every comparison
6. Fairness as arithmetic, and the choice you cannot avoid
7. Turning a complaint into a permanent test
8. Writing the result down
9. How much of this to actually do
10. Case study: Sixty days' notice and a set from January
11. Module test

Knowing If It Works: end-of-course examination

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

MODULE 1

What counts as evidence

Before any metric, you need a question worth answering and a set of examples worth trusting. This block builds the fixed, labelled, held-out set that every later lesson runs against: where the examples come from, how two people are made to agree on a label, what a synthetic example can and cannot stand in for, and what the number has to be compared against before it means anything.

> BY THE END OF THIS MODULE YOU CAN
>
> Turn a vague claim that a feature works into a fixed, labelled, held-out set with a stated agreement figure and a named baseline, and say what size of difference a set that size can and cannot detect

---

1 · 7 min

A demo is not a measurement

> THE ONE THING TO KEEP
>
> A result is a number, on a fixed set, next to a comparison — anything else is an anecdote.

The bot that worked perfectly on twelve messages

A small team in Lagos built a WhatsApp assistant that pulls delivery addresses out of customer messages. Before launch they pasted in twelve messages and read the outputs. All twelve were right. It looked good. They shipped.

In the first week the assistant handled about 3,000 messages and got roughly one in five wrong — not slightly wrong, but wrong in the way that sends a rider to the other side of the city. The twelve test messages had been written by the team, and the team writes addresses like "14 Adeola Odeku Street, Victoria Island". Customers write "opposite the blue mosque, after the second speed bump, call when you reach".

The model did not fail. The measurement failed. There was no measurement.

Three things "it looks good" cannot tell you

How often. Twelve passes feels like proof. It is not. If you see zero failures in twelve tries, a reasonable upper bound on your true failure rate is about three in twelve — one in four. A system failing a quarter of the time will still hand you twelve clean passes fairly often. Small informal samples can rule out "catastrophic". They cannot separate "very good" from "roughly okay", and that is exactly the distinction you need.

Compared to what. Better than the previous prompt? Better than a regular expression? Better than the cheaper model that costs an eighth as much? "Looks good" contains no comparison, so it cannot support a decision. Every real result is a difference between two things.

Read by whom. You are the worst possible reader of your own system's output. You know what the answer should be, so you fill in the gaps as you read. Hand the same output to someone who does not know the intended answer and watch how much of the "obviously fine" becomes "wait, which invoice is this about".

What a result actually looks like

A result has four parts: a number, the fixed set it was computed on, something to compare against, and enough version detail to reproduce it.

```
address-extract v3
  set:     address-eval-100  (frozen 2026-08-14)
  metric:  exact match on area + landmark
  score:   78%   (v2 on same set: 61%)   n=100
  model:   pinned id, temperature 0
```

That fits on four lines and it settles arguments. "I tried it and it seems better" starts them.

Why teams skip this

Not laziness. Three real reasons.

It feels slow. Building the set is an afternoon of unglamorous labelling while your colleague is shipping features. It feels premature. "We're still exploring, the prompt changes daily." That is precisely when you need it — you are making a change a day with no way to tell forward from backward. And it feels like admitting you might be wrong, because a number can disappoint you and a demo never will.

The afternoon pays for itself the first time two people disagree about whether a prompt change helped. Without a set, that argument is decided by seniority. With a set, it is decided in ten minutes.

The honest caveat

Evaluation does not make you right. It makes you *checkable*. A well-built eval set with a badly chosen metric will give you a confident number about something you do not care about, and the rest of this course is largely about that failure mode: measuring the wrong thing precisely.

Start anyway. A rough number on real examples beats a strong opinion on imagined ones, and you can improve a metric once you have one. You cannot improve a vibe.

BEFORE YOU MOVE ON

A team edits their support-reply prompt. They run the old version and the new version on the same 30 examples they have been using since January. Both versions produce a correct reply on all 30. What have they learned?

A. Very little: a set where both versions pass everything contains no example capable of telling them apart.
B. The new prompt is at least as good as the old one, since it did not lose on any example.
C. The change is not worth shipping, since it produced no improvement on the test set.
D. The problem is the 30 examples were hand-picked; 30 randomly sampled examples would have given a valid answer.

2 · 8 min

The decision the number has to serve

THE ONE THING TO KEEP

Write the decision, the threshold and the unacceptable failure before you see any result, or every number you get will look like agreement.

"Does it work" is not a question

A question is answerable when a specific answer would change what you do next. *Does the assistant work?* fails that test. Two people can read the same fifty outputs and disagree completely, because they are quietly answering different questions — one is asking whether this would embarrass us in public, the other whether it saves the support team an hour a day. Both are reasonable. Neither is the same measurement.

So before you pick a metric, write three lines. They take ten minutes and they save weeks.

The three lines

1. The decision. What will you actually do differently depending on the result? Ship or hold. Route all tickets through it or one in ten. Replace the human first draft, or keep the human and let the model suggest. If no decision hangs on the number, you are collecting it for decoration and you can stop now.

2. The threshold. What result would make you ship, and what result would make you not? State it as a number *before you see one*. "We ship if it pulls the correct amount and date from at least 90 of 100 invoices." A threshold written after the run is a rationalisation; every number looks like support for what you already wanted. A threshold written before is a commitment you can be held to, including by yourself at midnight.

3. The unacceptable failure. Most systems have one error that cannot be traded against anything else. A pharmacy assistant may not invent a dose. A refund bot may not promise money the policy does not allow. This does not belong in the average. It is a gate: score it separately, and let a single occurrence in a hundred fail the run, if that is genuinely the truth.

Three lines, dated, committed next to the eval set. That file is the closest thing applied AI has to a pre-registration, and it is the cheapest honesty device in this course.

A worked example

A clinic in Pune wants a WhatsApp bot that handles appointment requests. "Does it work" is unanswerable. Here is the same intention, written so it can be measured:

> ***Decision:*** *whether the bot answers messages directly outside office hours, or only drafts a reply for the receptionist to send in the morning.*

> ***Threshold:*** *on 100 real messages from last month, it extracts the requested doctor, date and time correctly in at least 92, and asks a clarifying question rather than guessing whenever any of the three is missing.*

> ***Unacceptable:*** *naming a doctor who does not work at the clinic, or confirming a slot that is not free. Zero tolerated in 100.*

Notice what changed. The vague version invited a demo. This version tells you what to collect, how many, what to label, and what result ends the argument.

Metrics that look like evidence and are not

Three keep appearing in decks:

- **Average user rating.** Response rates to thumbs-up widgets run at roughly one to two per cent of sessions, and the people who click are not a random sample. A 4.6 average tells you about the people who rate, not the people who use.
- **Tokens, sessions, messages.** Usage is not quality. A support bot whose message count doubled may be failing twice as often and being asked again.
- **A model's own confidence.** Ask a language model how sure it is and it will say 95%, often while wrong. Its stated confidence is text, produced the same way as the rest of the text, and it is not calibrated to anything.

The test for any proposed metric is one question: *if this number moved five points, what would we do?* If the honest answer is "nothing", it is not a metric, it is a mood.

Leading and lagging

The number you care about is usually slow. Tickets resolved without a human, refunds issued in error, hours saved per week — these take a month to move and are contaminated by everything else happening in the business.

So you keep two. A **lagging** metric that is the actual point, measured monthly and treated as the truth. And a **leading** metric on your fixed set, measured in ten minutes, which you believe *predicts* the lagging one. The relationship between them is a claim, and you should check it at least once: when the offline number moved and the real one did not, you have learned something more valuable than either number.

Say what would change your mind

The last line of the file is the useful one. Finish the sentence "we would abandon this approach if…". If nothing could produce that sentence, you are not running an evaluation. You are gathering support.

Most teams find this uncomfortable the first time, and most find that writing it makes the next three arguments shorter. The number is not there to prove you were right. It is there so that the room can stop guessing.

BEFORE YOU MOVE ON

A team is about to evaluate a summarisation feature. Which of these, written down before the run, does the most to keep the result honest?

A. A list of the eight metrics they will compute, so nothing important is missed.

B. A note that the model used is GPT-class and the temperature is 0.2, for reproducibility.

C. The number that would make them ship, the number that would make them stop, and the one failure they will not trade against anything.

D. A commitment to have two people read every output before drawing conclusions.

3 · 8 min

A hundred examples, made by hand

THE ONE THING TO KEEP

A hundred real labelled examples resolves fifteen-point differences and settles arguments; it cannot resolve three-point ones.

Where the examples come from

Real traffic, if you have it. Pull a week of production inputs, sample 100, and label them. That is the whole method and it beats every clever alternative.

If you have no traffic yet, you still have sources that are not your imagination: support tickets, the search box on your existing site, WhatsApp messages your sales person forwards you, the spreadsheet the ops team maintains by hand today. A used-car marketplace in Bogotá built its first eval set from 100 rows of the WhatsApp export its two agents had been answering manually for a year. Those 100 rows contained things nobody would have invented — customers sending voice-note transcriptions, prices written as "18 palos", three different spellings of the same neighbourhood.

Stratify, then label

Do not sample 100 items and hope the hard cases show up. Decide the slices you care about first, then fill each one:

- the common case, in proportion (usually 50 to 60 items)
- the known-hard cases: other languages, very long inputs, missing information, ambiguous requests
- the ones that already broke in production, every single one
- a few that *should* be refused — abuse, off-topic, requests you do not serve

Tag every item with its slice. Overall pass rate is the headline; pass rate per slice is where you actually learn something.

Keep it in the repository as JSONL, next to the code, in version control:

```
{"id":"addr-014",
 "input":"opposite blue mosque, after 2nd speed bump, ikeja",
 "expected":{"area":"Ikeja","landmark":"blue
mosque","street":null},
 "tags":["landmark-only","no-street-name"]}
```

Why a hundred

Because of what it can and cannot see. A 100-item set measures a pass rate to roughly plus or minus ten points. It will reliably show you a 15-point change. It will not show you a 3-point change, and you should stop pretending otherwise — teams routinely announce improvements far smaller than their set can resolve.

A hundred items gives about plus or minus ten points: enough to see a fifteen-point change, not a three-point one. Halving the interval costs four times the items.

So: 100 is enough to make decisions about prompt rewrites, model swaps, and "is this feature ready". It is not enough to tune. If you need to detect small differences, you need thousands of items, which means automated labelling, which means the next lesson's problems. Most teams never need that.

A hundred items also takes about three hours to label properly. Three hours is a price you can pay this week.

Two people, twenty items

Before you label the rest, have a colleague independently label the same 20 items. Then compare.

If you disagree on 6 of 20, your *criterion* is broken, not the model. Two competent people cannot agree on what "a helpful reply" means, so no model can be scored against it and no judge can automate it. Rewrite the criterion until two humans agree at least 90% of the time, then label the rest. This single step catches more bad evaluation than anything else in this course.

Freeze it, then grow it

Once labelled, the set is frozen. You do not edit it to make a number look better. When production produces a failure your set did not predict, that failure becomes a new item — this is how the set earns its keep over a year.

One legitimate reason to touch existing items: your own labels were wrong. That happens, often to 5 or 10 items in a first set. Fix them, write down what you changed, and re-run every previous version against the corrected set so your history stays comparable. A corrected label with a note is honest. A quietly deleted hard item is how a set slowly becomes a set of things you already pass.

BEFORE YOU MOVE ON

Reviewing a first eval run, you find eight items where the model's answer was right and your own "expected" answer was wrong. What is the sound move?

A. Correct the eight labels, note the change, and re-run the earlier prompt versions so the old and new numbers still compare.

B. Leave them as they are — changing the set after seeing the results is exactly how people fool themselves.

C. Remove those eight items, since ambiguous cases only add noise to the score.

D. Keep the labels and mark those eight as passes for this run, since the model was in fact right.

4 · 9 min

Where the examples come from

THE ONE THING TO KEEP

Sample your traffic deliberately — stratified, deduplicated, and weighted back to real proportions — because the easiest hundred items to grab are the least representative hundred you own.

The last hundred requests are not a sample

The quickest way to build an evaluation set is to open the logs, copy the most recent hundred rows, and start labelling. It feels like real data because it is real data. It is still a bad sample, for four reasons you can name before you look.

- **Recency.** The last hundred rows cover roughly the last twenty minutes. If you pull them at 11am on a Tuesday you have a working-hours, weekday, one-time-zone sample.
- **Volume bias.** One integration customer running a nightly script can be forty of your hundred rows. You will tune the system for that script.
- **Survivor bias.** Logs contain what people asked a system that already exists. Questions your product visibly cannot answer stopped being asked months ago.
- **Head dominance.** Most traffic is a handful of intents. A uniform sample of 300 from 40,000 weekly queries might contain two refund requests, which is not enough to measure anything about refunds.

Draw the sample on purpose

Decide the strata first — the dimensions along which the system plausibly behaves differently. For a support assistant that is usually intent, language, and input length. Then sample within each stratum.

```
# 25 items per intent, reproducible, from a week of logs
sample = (df
          .drop_duplicates(subset="normalised_text")
          .groupby("intent", group_keys=False)
          .apply(lambda g: g.sample(n=min(25, len(g)),
random_state=0)))
```

This is stratified sampling, and it buys you the ability to say something about refunds. It also breaks your headline number, and you have to fix that deliberately.

Weight back, or do not quote a headline

If refunds are 1% of traffic and 12% of your set, then a set-wide pass rate is answering a question about a world that does not exist. Two honest options:

1. **Report per stratum and never average.** "94% on billing, 71% on refunds, 88% on account changes." Most useful in practice.
2. **Weight back to production proportions.** Multiply each stratum's rate by its real traffic share and sum.

```
overall = sum(rate[s] * traffic_share[s] for s in strata)
```

The weighted number has wider uncertainty than its item count suggests, because the rare strata carry heavy weights on few items. Say so when you quote it.

Deduplicate before you label, not after

Near-duplicates are the silent inflater. Four hundred copies of the same automated message make a set that looks like 400 items and behaves like one. Normalise (lowercase, strip identifiers and timestamps), hash, and keep one — but record how often it occurred, because that frequency is exactly what you need for the weighting above.

The check that catches this: if any single normalised string is more than 2% of your set, it is not a set, it is a template.

Real user text is personal data

Logs are people's words. Before any of it lands in an evaluation file:

- Redact identifiers automatically first — emails, phone numbers, account numbers, addresses — and then have a human read the sample, because automatic redaction misses the sentence where somebody names their employer.
- Check what your privacy notice actually promised, and keep the set inside whatever retention window applies.
- Store it where the rest of your production data lives, not in a spreadsheet on a laptop. An eval set is a copy of your users' most sensitive text with the access controls removed.

Where policy forbids using real content at all, you fall back to team-written and synthetic items, and you say in the report that the set is not drawn from traffic. That is a real limitation, not a footnote.

When you have no logs at all

A feature that has not shipped has no traffic. You then have three sources, in descending order of trustworthiness:

1. **Adjacent real text** — support tickets, search queries, forum posts about the same job, even if they were never typed at your system.
2. **Team-written items**, produced by people who have watched users work. Have two people write independently and compare; the overlap is the obvious case, the difference is the interesting one.
3. **Synthetic items**, which are the next lesson and come with the largest caveats.

Write the provenance of every item into the file — `source: logs | adjacent | team | synthetic` — because six months later nobody remembers, and every argument about a surprising number starts with where the items came from.

The habit

Sample once a quarter, not once. Traffic moves: a marketing campaign, a new market, a competitor's outage, a festival. A set drawn in March and still quoted in November is measuring a product that no longer exists. Keep the March set frozen for comparability, and add a fresh sample beside it — the gap between the two is itself a measurement.

BEFORE YOU MOVE ON

A team stratifies its eval set so that refunds, which are 1% of live traffic, make up 12% of the 300 items. They report a single set-wide pass rate of 86%. What has gone wrong?

A. Nothing — stratifying makes the set more representative, so the average is more trustworthy than a uniform sample would have been.

B. The 86% describes a world where 12% of requests are refunds, so it overstates or understates live performance depending on whether refunds are harder.

C. Stratified samples cannot be used for pass rates at all, only for precision and recall.

D. The set is too small for a headline number, and 3,000 items sampled the same way would fix it.

5 · 9 min

Writing a labelling guide two people can agree on

THE ONE THING TO KEEP

Two people labelling the same twenty items blind, and a written ruling for every disagreement, is what turns an opinion into a criterion.

The guide is the real artefact

When you label a hundred examples, the labels are not the valuable output. The document that made two people produce the *same* labels is. That document is what lets you add 100 more items next quarter, hand the task to somebody new, or write a judge prompt in module three. Without it, your set is a private opinion that decays as your memory of it fades.

A labelling guide is short. One page per label. It contains:

1. **A definition in one sentence**, stated as something observable in the output rather than something felt about it.
2. **Three examples that pass**, copied verbatim from real data.
3. **Three that fail**, with the specific span that fails it underlined in words.
4. **Two boundary cases with a ruling** — the ones you argued about, and the decision you made, even if the decision was arbitrary.

The fourth item is the one people skip and the one that does the work. Boundary rulings are how a criterion survives contact with data nobody imagined.

From adjective to observable

Most first drafts are unlabellable:

> *The reply should be helpful and professional.*

Neither word points at anything in the text. Rewrite until each rule names something you could highlight with a cursor:

> *- The reply answers the question that was asked, rather than a related one.*

> *- The reply contains no commitment about dates, money or eligibility that is absent from the policy text supplied.*

> *- The reply is in the same language as the customer's message.*

> *- If required information is missing, the reply asks for it instead of assuming a value.*

Four rules, each separately true or false. "Helpful and professional" is now four columns in a spreadsheet, and a disagreement can be located: you and your colleague differ on rule two, not on the vibe.

Measure agreement, and do not be fooled by the raw number

Have a second person label the same 20 items independently. Then compute agreement — but be careful which agreement you compute.

Suppose 94 of every 100 support replies are fine. Two labellers who both mark everything "fine" agree 94% of the time and have learned nothing. Raw agreement flatters you exactly when the classes are imbalanced, which is most of the time in safety and quality work.

Cohen's kappa corrects for the agreement you would get by chance. Roughly:

```
kappa = (observed_agreement − chance_agreement)
        / (1 − chance_agreement)
```

In the example above, chance agreement is already about 0.89, so 94% raw becomes a kappa near 0.45 — poor. Useful reference points, and they are conventions rather than laws:

- **0.8 and up** — the criterion is solid; automate against it.

- **0.6 to 0.8** — workable, but every measurement you take carries that much slack.
- **below 0.6** — you do not yet have a criterion. Stop labelling and rewrite the guide.

Three lines of Python get you there: `sklearn.metrics.cohen_kappa_score(a, b)`. If you would rather not install anything, the formula above runs fine in a spreadsheet.

Disagreements are data, not friction

When the two of you differ on six of twenty, sit down and go through the six. Each one resolves into exactly one of three things:

- **The guide is silent.** Add a boundary ruling. This is the good case and it is most of them.
- **One person misread the item.** Fine. No change needed.
- **The two of you actually want different things from the product.** This is the expensive discovery, and it is far cheaper found now than after launch. It is not a labelling problem at all; it is a product decision wearing a labelling problem's clothes.

Re-label a fresh 20 after each guide revision. Two rounds is usually enough. Three means the criterion is genuinely hard, and you should consider splitting it into two simpler ones.

Label noise is a ceiling

Here is the consequence people miss. If your labels are themselves 92% correct, then a perfect model scores 92% on your set — and worse, a model that has learned your labellers' particular mistakes scores *higher* than one that is right. No amount of prompt engineering climbs above the quality of the ruler.

So when a score plateaus around 90%, the next honest step is not another prompt variant. It is to take the 30 items the system failed, re-label them blind, and find out how many of those "failures" were your own labels being wrong. Teams routinely find a third.

Tools, including the free ones

At 100 items, a spreadsheet with one row per item and one column per rule is not a compromise; it is the right tool, and it exports to CSV. Past a few hundred items, or with more than two labellers, **Label Studio** and **Argilla** are both open source, self-hostable and free, and they give you the two things a spreadsheet does not: blind labelling, so nobody sees the other person's answer, and a built-in agreement report. Commercial platforms such as Scale or Labelbox do the same job with a sales call attached.

Whatever you use, keep the guide in version control beside the data, and put the date on it. A guide with no history is a guide nobody can trust six months later — including you.

BEFORE YOU MOVE ON

Two reviewers independently label 200 support replies for a rare policy violation. They agree on 191 of them, which sounds excellent. Only 12 of the 200 were actually violations. What should they conclude?

A. Agreement is high enough to proceed; 95.5% is comfortably above any reasonable bar.

B. Because violations are rare, most of that agreement is chance; they should compute kappa, which will be far lower, before trusting the criterion.

C. The nine disagreements are noise from reviewer fatigue and can be resolved by whichever reviewer is more senior.

D. They need a larger sample, since 200 items give too few violations to measure anything.

6 · 9 min

How much your labellers actually agree

THE ONE THING TO KEEP

Raw agreement flatters you whenever one class is common, so compute kappa as well — and read a low kappa as a fault in the definition rather than in the people.

Ninety-two per cent agreement can be worth almost nothing

Two people label 100 posts as harmful or fine. They agree on 92. That sounds like a settled definition.

Now look at the marginals. Labeller A called 12 posts harmful, labeller B called 10. Both spend most of their time saying "fine", and two people who both say "fine" nine times out of ten will agree a lot by accident alone.

Two labellers, 100 posts, 92 agreements

	B: harmful	B: fine
A: harmful	**7** both harmful	**5** A only
A: fine	**3** B only	**85** both fine

Agreement is 7 + 85 = 92. But two people who both say 'fine' about nine times in ten would agree on roughly 80 posts by accident alone, which is why kappa comes out at 0.59 rather than 0.92.

Cohen's kappa measures how much of the agreement is left once you subtract the accidental part.

```
p_o = observed agreement            = 0.92
p_e = agreement expected by chance
    = (0.12 * 0.10) + (0.88 * 0.90)
    = 0.012 + 0.792                 = 0.804

kappa = (p_o - p_e) / (1 - p_e)
      = (0.92 - 0.804) / 0.196      = 0.59
```

Ninety-two per cent agreement is a kappa of 0.59. The conventional bands — fair 0.21 to 0.40, moderate 0.41 to 0.60, substantial 0.61 to 0.80 — put that at the top of "moderate". These bands are a 1977 convention by Landis and Koch, not a law of nature; quote them as a shared vocabulary, never as a pass mark.

In Python that is one line, and it is free:

```
from sklearn.metrics import cohen_kappa_score
cohen_kappa_score(labels_a, labels_b)          # 0.59
```

If you have no Python, the four counts and the formula above fit in a spreadsheet in five minutes.

The paradox that will confuse you at some point

Push the imbalance further. Suppose only 2% of posts are harmful, and the two labellers agree on 95 of 100 items. Chance agreement is now around 0.94, so kappa collapses towards zero even though the raw agreement went up. This is the well-known kappa paradox: when one class dominates, `p_e` approaches `p_o` and the ratio becomes unstable.

The mechanism matters more than the name. Kappa is asking "how much better than guessing at these rates", and when guessing is already 94% correct there is very little room left to be better in. So:

- Always report the raw agreement, kappa, and the two marginals together. Any one of them alone can mislead.
- On very rare classes, prefer agreement measured only on the positives — of the items either person flagged, how many did both flag. That is the

Jaccard overlap, and it does not have a chance-correction term to blow up.

Low kappa is a fault in the guide

The instinct on seeing 0.31 is to retrain the labellers. That is almost always the wrong move. Print the disagreements and read them. They cluster, and usually into two or three causes:

- **A missing rule.** Nobody said what to do with quoted abuse, so one person removes and the other allows.
- **A hidden dimension.** Half the disagreements are in Hinglish, where one labeller reads a word as an insult and the other as ordinary slang.
- **A scale nobody can hold.** A 1 to 5 quality score with no anchors becomes each person's personal mood. Binary questions agree far better than scales, which is why a rubric of several binary checks beats one holistic rating.

Fix the guide with a written ruling for each cause, then relabel the affected items. Kappa is a diagnostic instrument pointed at your definition.

More than two people, or a scale

Cohen's kappa handles exactly two raters on categories. Beyond that:

- **Fleiss' kappa** for three or more raters on categories.
- **Krippendorff's alpha** for any number of raters, missing labels, and ordinal or interval scales. The `krippendorff` package on PyPI is free and about four lines to use.
- **Weighted kappa** when adjacent categories are nearly-agreements — 4 versus 5 on a five-point scale should not be punished as hard as 1 versus 5.

Why this number governs everything downstream

Two consequences follow immediately, and both come up later in the course.

First, the labels are your ground truth, and their noise sets a limit on any measurement made against them. If two careful people agree at kappa 0.6, a mod-

el scored against one person's labels cannot be meaningfully separated from a model scored against the other's when the gap between models is small.

Second, when you build an automated judge, the question is never "does the judge agree with a human". It is "does the judge agree with a human about as well as two humans agree with each other". Without the human-human figure, the judge's agreement score has no scale to sit on. Compute this number early, write it at the top of the labelling guide, and re-compute it whenever the guide changes.

BEFORE YOU MOVE ON

A safety label is applied to 2% of items. Two reviewers agree on 96 of 100 items, and Cohen's kappa comes out at 0.09. What is the most defensible reading?

A. The reviewers are unreliable and the labelling task should be abandoned until they are retrained.

B. Kappa is the wrong statistic for this task, so raw agreement of 96% should be reported instead.

C. Chance agreement is already about 96%, so there is almost no headroom for kappa to measure, and agreement should be judged on the flagged items only.

D. The two reviewers disagreed on four items, which at this base rate is roughly two items' worth of real disagreement, so the definition is fine.

7 · 8 min

Test cases you generate, and what they cannot tell you

THE ONE THING TO KEEP

Generated items measure whether a system can handle a shape of input; they cannot tell you how often that shape occurs, so they belong in a capability suite and never in the headline number.

Why anybody reaches for this

There are three honest reasons to generate test items rather than collect them: the feature has not shipped and there are no logs; the case you care about is real but rare, so a natural sample contains two of them; or privacy rules forbid copying user text into an evaluation file at all.

There is also a dishonest reason, which is that generating 500 items takes ten minutes and labelling 100 takes two days. That one produces sets that look impressive and measure nothing.

Two kinds of generation, and one is much safer

Templates with slots are deterministic. You write the shapes and enumerate the values.

```
TEMPLATES = ["cancel my {product} subscription before {date}",
             "{product} ka subscription band karna hai {date}
se pehle"]
PRODUCTS  = ["pro", "family", "annual plan"]
DATES     = ["31 March", "31/03", "next Tuesday", "aaj raat"]

cases = [t.format(product=p, date=d)
         for t in TEMPLATES for p in PRODUCTS for d in DATES]
```

You know precisely what varies, so a failure localises: it fails on `31/03` and not on `31 March`, which is a date-parsing bug you can hand to somebody. Templates are boring and they are the most useful synthetic data most teams ever make. Free tools are enough — Python's `itertools` and the `Faker` package generate names, addresses and phone numbers in dozens of locales.

Model-generated variation gives you fluency and surprise: "rewrite this request forty ways, as an angry customer, a terse one, one typing on a phone with autocorrect". It reads like real traffic. It is also the risky kind, for a specific mechanical reason.

The blind-spot mechanism

A language model generates text from its own high-probability region. Ask a model to invent hard cases for a system built on that same model family, and it produces the difficulties it can already represent — which are disproportionately the ones it handles. The failures you most need are the inputs the model would not think to write, because not thinking to write them and not handling them come from the same gap in the model.

You do not have to take this on faith. Measure it. Run the system over 100 real items and 100 generated ones, both labelled the same way.

- If the generated set's pass rate is within a few points of the real one, it is a usable stand-in for that slice.
- If it is fifteen points higher — the common result — your synthetic set is easier than reality and cannot carry a headline number.

Record that gap in the report. It is one number and it converts an argument into a fact.

What generated items are genuinely good for

- **Structural coverage.** Empty input, 20,000 characters, emoji-only, mixed scripts, right-to-left text, code-switching between Hindi and English mid-sentence, a CSV pasted into a chat box. Real traffic contains all of these and your sample of 300 contains none of them.

- **Controlled variation.** The same request with the amount changed from 100 to 100,000 to -100, to see where the behaviour breaks.
- **Adversarial seeds.** A starting list of injection and jailbreak attempts, which you then extend by hand. The safety lesson later in the course treats this properly.
- **Privacy-safe stand-ins.** Same shape, invented content, so the file can live somewhere with fewer restrictions.

What they cannot do

Synthetic items answer "can it", never "how often". Frequency is a property of your users, and no generator has access to your users. This one distinction settles most disputes about whether a synthetic set is acceptable:

- "Can the system handle Devanagari input?" — a generated set answers this.
- "What fraction of our failures come from Devanagari input?" — only sampled traffic answers this.

So keep two files. A **capability suite** of generated and hand-written items, which you run as a pass or fail gate and which is allowed to be unrepresentative on purpose. A **traffic set** of sampled real items, which produces the number you quote to anybody outside the team. Tag every item with its `source` field, and make the harness refuse to average across sources.

The failure to watch for

Generated sets rot in a particular way: somebody adds items whenever a model does something amusing, the file grows to 4,000 items, nobody has read most of them, and a third of them are near-duplicates of each other. Then a change that fixes one template appears to fix eighty items and the score jumps six points for no reason.

Deduplicate on normalised text, cap the number of variants per template, and review the file the way you would review code. A test file nobody reads is not a test.

BEFORE YOU MOVE ON

A team generates 400 test questions with a model, measures 91% correct, and reports it as the system's accuracy. Real traffic later shows 74%. What is the mechanism behind the gap?

A. The generated set was too small, and 4,000 generated items would have converged on the true figure.

B. Generated questions come from the model's own high-probability region, so they over-represent the phrasings the system handles and carry no information about how often hard inputs occur.

C. The generated questions were probably mislabelled, since nobody checked the expected answers by hand.

D. Real traffic contains adversarial users, and the generated set contained none, which accounts for the whole difference.

8 · 9 min

Two sets, and the ways they get contaminated

THE ONE THING TO KEEP

Keep a development set you tune against and a test set you almost never look at, split by group and by time, because a set you have optimised against no longer predicts anything.

Why one set is not enough

You will look at your evaluation set constantly. You will read the failures, change the prompt to fix them, re-run, read the new failures, change it again. After twenty rounds of that, the score on that set is no longer an estimate of how the system behaves on new inputs. It is a record of how well you fitted a hundred specific examples, using your own brain as the optimiser.

This is overfitting, and it does not require a training loop. It happens to prompts, to retrieval configurations, to threshold choices, to anything you tune while watching a number.

The fix is two sets, with different rules:

- **The development set.** Look at it as often as you like. Read every failure. Tune against it. This is where the work happens.
- **The test set.** Do not read the items. Run it when a decision is being made — before a release, before telling somebody a number — and then leave it alone. Once a month is generous. Once a quarter is common.

Split them before you start, by a rule that cannot drift: a hash of the item id, `int(md5(id)[:8], 16) % 10 < 2` for a 20% test set. Written that way, adding new items later keeps the split stable and nobody has to remember anything.

If the test number is much worse than the dev number, you have not been improving the system. You have been improving your fit to the dev set. That gap is the most useful diagnostic in this course and it is invisible with one set.

Near-duplicates are the commonest leak

Suppose your data comes from support tickets. "How do I reset my password" appears 340 times in slightly different wording. Sample randomly into two sets and near-identical items land on both sides. Your test set is now measuring the same question your dev set already taught you to answer, and it will look wonderful.

One team building an FAQ router reported 94% on a held-out set and 61% in production. Thirty of their hundred held-out items were paraphrases of items in the training prompt's few-shot examples. The set was not testing generalisation at all.

Dedupe before splitting. A cheap and surprisingly effective pass: normalise (lowercase, strip punctuation, collapse whitespace), then drop exact hash matches; then embed everything with a small local model such as `all-MiniLM-L6-v2` through `sentence-transformers` — free, runs on a laptop

CPU — and drop anything with cosine similarity above about 0.95 to an item already kept. Look at twenty of the pairs you dropped before trusting the threshold.

Split by group, not by row

Rows are usually not independent. Twelve tickets from the same customer share vocabulary, product, and often the same underlying bug. Ten pages from the same document share formatting. Put some in dev and some in test and information crosses the wall.

So split by the unit that shares information: customer, document, seller, conversation, patient. All rows for one group go entirely to one side. `GroupKFold` in scikit-learn does this in one line, and doing it by hand in SQL is a `GROUP BY` and a hash.

Split by time when time matters

If what you are building will run on next month's data, evaluating on a random sample of last year's is optimistic. Language, product names, scam patterns and customer questions all move. Take the last four weeks as the test set and everything earlier as dev. The number will be lower than a random split gives you. The lower number is the honest one.

This also catches an error that random splits hide entirely: features computed with information that would not exist at prediction time. If your input includes a field the system only learns *after* the event you are predicting, a random split rewards it and a time split exposes it.

Benchmark contamination

There is a version of this problem you cannot fix by splitting, because the leak happened before you arrived. Public evaluation sets — the ones a model's card quotes — are on the open web, and models are trained on the open web. A model may have seen the questions and the answers.

The signs are recognisable: near-perfect scores on an old public benchmark alongside ordinary performance on a freshly written variant of the same task;

sensitivity to trivial rewording that should not matter; occasionally, verbatim recall of a question's phrasing. Some benchmark authors embed *canary strings* — unique identifiers that should never appear in training data — so contamination can be detected after the fact. It usually is not checked.

The practical consequence is narrow and important: your own set, made from your own data, after your model's training cutoff, is the only set you can be confident is uncontaminated. That is a large part of why lesson two asks you to build one by hand.

The rule that keeps it working

Write the rules down where the set lives: which split is which, who may look at what, and how often the test set is run. Then treat a test-set peek the way you would treat a schema migration — not forbidden, but noticed, logged and deliberate.

The first time somebody quietly tunes against the test set to make a release, the set stops being evidence, and nothing in the output tells you that it has happened.

BEFORE YOU MOVE ON

A team building a document-classification feature splits 4,000 pages randomly into dev and test. Both sets score around 93%, but live accuracy is near 70%. The pages came from 180 source documents. What is the most likely explanation?

A. The 4,000 pages are too few to estimate accuracy, so both numbers are noise.

B. The test set was run too often, so the team has overfitted to it as well as to dev.

C. Live traffic is simply harder than the collected data, and nothing about the split is at fault.

D. Pages from the same document landed on both sides of the split, so the test set was measuring documents the system had effectively already been tuned on.

9 · 9 min

The number above it and the number below it

THE ONE THING TO KEEP

A score is only readable between a trivial baseline and the human agreement ceiling, and most impressive-looking results shrink to two or three points once both are drawn.

Seventy-eight per cent of what

An intent router is reported at 78% accuracy. Nobody in the room can tell whether that is good, and the argument that follows is about vibes.

Four numbers make it readable, and all four are cheap.

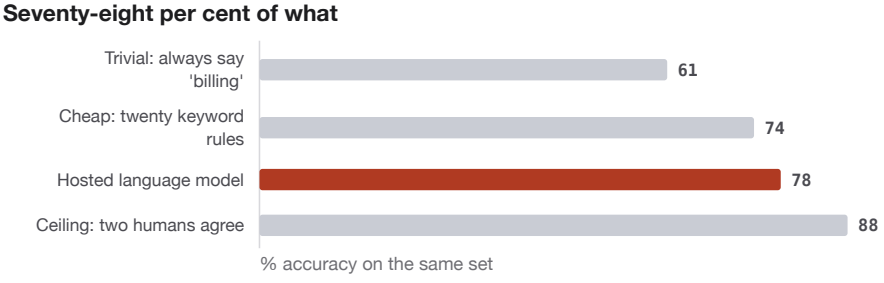
The trivial baseline. Always predict the most common class. If 61% of tickets are billing, the do-nothing system scores 61%. Your model's real contribution is 17 points, not 78.

The cheap baseline. Twenty lines of keyword rules, or a bag-of-words logistic regression trained in a second on the same labels. On routing tasks this often lands at 72 to 76%. Now the language model's contribution is two to six points, at a per-call cost the regex does not have.

The incumbent. Whatever the system replaces — the old model, or the human process. If agents currently route correctly 85% of the time, an 78% model is a regression dressed as a launch.

The ceiling. Two people labelling the same items agree, say, 88% of the time. Above that, "accuracy" is mostly measuring which annotator you happened to score against.

Written out, the result is not "78%". It is: *trivial 61, keywords 74, model 78, humans 88*. That sentence takes ten seconds to read and settles the decision.



The model adds 17 points over the trivial baseline and four over keyword rules, bought with 900 ms of latency and a vendor in the failure path. Above 88 the labels themselves disagree.

Building them properly

Free tools do all of this. `scikit-learn` has the trivial baselines built in:

```
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

DummyClassifier(strategy="most_frequent").fit(X, y).score(Xte,
yte)   # 0.61
make_pipeline(TfidfVectorizer(),
LogisticRegression(max_iter=1000)
              ).fit(X, y).score(Xte, yte)
# 0.74
```

For retrieval, the equivalent is BM25 — keyword scoring that has been the strong classical baseline for thirty years. The `rank_bm25` package runs it on a laptop with no GPU. On corpora full of product codes, part numbers and jargon it frequently comes within a few points of an embedding model, for the mechanical reason that exact term matching is unbeatable on rare exact terms, which is precisely what a vector average smooths away. Teams that skip this baseline spend a quarter building a vector database to gain three points they could have measured in an afternoon.

Spend an hour on the baseline, not five minutes. A badly built baseline flatters you, and everyone downstream inherits the flattery. Write down its configuration next to its score, the same way you would for the model.

The ceiling is not 100%

A great deal of wasted effort comes from treating perfect as the target on tasks where the labels themselves are contested. If two careful annotators agree at 88%, then:

- A model at 86% is roughly at the level of a second human. Further gains are partly fitting one annotator's idiosyncrasies.
- The way to move past it is not a better model. It is a better definition — sharper rules, split categories, or a task decomposed into questions people can agree on.

The previous lesson's kappa number is what makes this concrete. Compute it once, and it stops the "why are we not at 95%" conversation permanently.

Baselines have costs, and that is part of the comparison

Report cost and latency for the baseline too, because that is where the argument actually ends. A realistic line for a routing feature:

- keyword rules: 74%, 0.4 ms, no per-call cost, no external dependency
- small local model: 76%, 40 ms, no per-call cost, runs on a CPU
- hosted large model: 78%, 900 ms, 0.2 cents per call, a vendor in the failure path

Four points cost you nearly a second of latency and a dependency that can change under you. Sometimes worth it — an assistant that answers ambiguous free text often is. Sometimes plainly not. The point is that the decision becomes visible instead of being made by whoever spoke last.

The baseline nobody writes down

For a product, there is one more: **not having the feature at all**. Some AI features are measured against a fantasy of what users wanted rather than against the plain link, the search box, or the form that already worked. If a summary of a document is right 82% of the time and the document is three paragraphs long, the honest comparison is with reading the document.

Ask it explicitly, before any of the arithmetic: what happens if we ship nothing here. Then measure against that.

BEFORE YOU MOVE ON

A classifier for a rare compliance breach reports 97% accuracy on a set where 4% of items are breaches. Which single figure most changes how you read it?

A. The size of the test set, since 97% on 100 items has an error bar of roughly eight points.

B. The score of a model that always predicts "not a breach", which is 96% on this class balance.

C. The F1 score, which combines precision and recall into one comparable number.

D. The inter-annotator agreement, since that sets the ceiling the model is being measured against.

---

10 · 9 min

How wrong your number probably is

THE ONE THING TO KEEP

A pass rate on 100 items carries roughly eight points of uncertainty, but comparing two systems on the same items — counting which ones flipped — can detect far smaller differences.

A rate from a sample is an estimate

You ran 100 items and 82 passed. The system's pass rate is not 82%. Eighty-two per cent is your best guess at it, drawn from one sample of a hundred, and if you drew a different hundred you would get a different number. How different is something you can calculate in one line, and almost nobody does.

For a pass rate `p` measured on `n` items, the standard error is:

```
se = sqrt(p * (1 - p) / n)
```

At p = 0.82 and n = 100 that is 0.038. A 95% interval is roughly plus or minus two standard errors, so about **±8 points**: your true rate is plausibly anywhere from 74% to 90%.

Sit with that for a moment, because it reframes most of the conversations you have had about model quality. On a hundred items:

- 82% versus 79% is nothing. The intervals overlap almost entirely.
- 82% versus 91% is probably real.
- Announcing an improvement from 82% to 84% is announcing noise.

And the arithmetic runs the other way too. To halve the interval you need four times the data: ±8 points at 100 items becomes ±4 at 400 and ±2.5 at 1,000. Precision is expensive and you should buy only as much as your decision needs.

Paired comparison is far more sensitive

Here is the part that rescues small sets. When you compare two prompts, you are not comparing two independent samples. You are running **the same items** through both. That pairing carries information, and using it can detect differences the naive interval says are invisible.

Forget the two totals. Count the four cells:

- both passed — tells you nothing about the difference
- both failed — tells you nothing either
- old passed, new failed: call it `b`
- old failed, new passed: call it `c`

Only `b` and `c` carry the comparison. If the change did nothing, `b` and `c` should be about equal. McNemar's test formalises this, and a usable rule of thumb is that the difference is worth believing when:

```
abs(b − c) > 2 ∗ sqrt(b + c)
```

An example. Old prompt 82/100, new prompt 88/100. Six points, well inside the ±8 interval, so on the naive reading you cannot conclude anything. But suppose the flips are `b = 2` and `c = 8`. Then `abs(b − c) = 6` and `2 ∗ sqrt(10) = 6.3` — just short, so still not conclusive, but nearly. Whereas if the flips are `b = 1, c = 7`, you have `6 > 5.3` and a real result on a hundred items.

Same 100 items: old prompt 82 passed, new prompt 88

	New: passed	New: failed
Old: passed	**81** no information	**1** b: regressed
Old: failed	**7** c: improved	**11** no information

Only b and c carry the comparison. Here $|b - c| = 6$ against $2\sqrt{(b + c)} = 2\sqrt{8} \approx 5.7$, so this six-point gap is real on a hundred items, even though the naive ±8 interval would have called it noise.

The same 100 items, the same 6-point gap, and a completely different conclusion depending on *which* items moved. This is why lesson five insists on tracking flips rather than totals, and it is the single highest-value statistical idea in the course.

The bootstrap, for everything that is not a simple rate

Mean latency, median cost, average judge score, recall@5 — none of these have a neat formula you will remember. The bootstrap gives you an interval for any statistic without one, and it is five lines:

```
import numpy as np

scores = np.array(per_item_scores)   # one number per eval item
boot = [np.mean(np.random.choice(scores, len(scores),
replace=True))
        for _ in range(2000)]
low, high = np.percentile(boot, [2.5, 97.5])
print(f"{scores.mean():.3f}  95% CI [{low:.3f}, {high:.3f}]")
```

Resample your own items with replacement, recompute the statistic, do it two thousand times, take the middle 95% of the answers. That is the whole idea. It works for any statistic you can compute from per-item results, which is why storing per-item results — not just the summary — matters so much.

For a paired comparison, bootstrap the *difference*: resample the item indices once, compute both systems' scores on that resample, take the gap, repeat. If the resulting interval crosses zero, you cannot call it.

Non-determinism is a second source of spread

All of the above assumes one run. At temperature above zero, the same input gives different outputs, so there is variance in the system as well as in the sample. Run your set three times and you may see 82%, 79% and 84% with nothing changed at all.

The honest response is to run the set three or five times and report the mean and the spread across runs. If run-to-run spread is 5 points, then a 4-point improvement is not detectable no matter how many items you add. Setting temperature to 0 reduces this but does not eliminate it — batching, load and provider-side changes all leak in.

Write the number the honest way

One line, always in this shape:

> *Pass rate 82% (95% CI 74–90, n = 100, 3 runs, spread 3 pts), against 79% (74–86) for the previous prompt on the same items; 8 items improved, 5 regressed.*

It is longer than "82%" and it is the difference between a measurement and a claim. It also has a social effect worth having: once a team writes intervals, it stops shipping on three-point movements, and the arguments about whose change helped get much shorter.

BEFORE YOU MOVE ON

You compare two prompts on the same 100 items. Old scores 74, new scores 80. Of the items, 3 went from pass to fail and 9 from fail to pass. What is the soundest reading?

A. The flip counts support a real improvement: 9 versus 3 is a bigger gap than the paired rule of thumb tolerates by chance.

B. Six points is inside the ±9 point interval for a rate near 0.77 on 100 items, so the result is inconclusive.

C. The 3 regressions cancel the improvement, so the net gain is only 6 items and cannot be distinguished from noise.

D. Nothing can be said until the set is grown to at least 400 items.

CASE STUDY · MODULE 1

Fifteen questions and a batch starting Monday

A physics coaching centre in Kota, two weeks before the new JEE batch, deciding whether an AI doubt-solver is ready

Vidyarthi Classes has 1,400 students across three batches and a doubt-solving problem that every coaching centre has: between nine at night and seven in the morning, nobody answers. The founder, Rakesh, had been paying two junior teachers to sit on the WhatsApp doubt groups until midnight, and they were leaving.

Over the summer a former student, now in his second year at an engineering college, built a doubt-solver on top of a hosted language model. You send a photo or a typed question, it replies with a worked solution. Rakesh tried it himself. He typed fifteen questions from the printed module on kinematics and rotational motion, read the answers, and all fifteen were right. Two were better explained than the printed solutions. He wanted it in every group by Monday, when the new batch started, because a competitor two streets away had started advertising "24x7 AI doubt support".

The head of physics, Mrs Iyer, had been teaching for twenty-two years and did not trust it. Not on principle: she had read three of the fifteen answers and thought they were fine. Her objection was that she had no idea what the other few thousand answers would look like. "The questions you typed are from our own module. Our students do not send those. They send a blurry photo of a question from a random book, with half the diagram cut off, and they write 'sir why not option C'."

Rakesh's position was that fifteen for fifteen was evidence, and that the cost of waiting was real. Every night the tool was not live was a night the competitor's advertisement was true and his was not. He estimated that three days of building a proper test would cost him the launch date, and that a fortnight of "it works, we tried it" was worth more than a number.

Mrs Iyer's position was that fifteen for fifteen told them almost nothing. If the true failure rate were one in four, fifteen clean passes would still happen often

enough that nobody should be surprised. She also pointed out that Rakesh was the worst possible reader of the outputs, because he knew what the answer should be and filled in the gaps as he read. And she had a specific fear: a solver that confidently gives a wrong numerical answer to a student at 1 a.m., who then goes into the test believing it.

The two of them agreed on one thing, which was that the argument was going in circles. So they wrote down what a decision would actually require.

The decision was not "launch or not". It was whether the tool answered students directly in the groups, or whether it drafted an answer that a junior teacher approved in the morning. Mrs Iyer wanted a threshold: on a hundred real doubts from last year's groups, the tool had to give a correct final answer on at least 85, and for numericals the answer had to be correct to the printed solution. The unacceptable failure was a confident wrong numerical answer: she wanted that scored separately, and she wanted a single occurrence in a hundred to fail the run.

Building the set was the cost. The doubt groups from last year were still on two teachers' phones. Pulling a hundred real doubts, stratified so that photographed questions, typed questions, conceptual doubts and numericals were all represented, would take a day. Labelling them took two teachers, and the labels had to agree with each other, which meant a labelling guide and a check that the two teachers actually applied it the same way. Three days, two senior teachers, at the busiest time of the year.

Rakesh's counter-offer was to launch in one batch on Monday and "watch it closely". Mrs Iyer's counter-offer was that watching it closely was what he had just done with fifteen questions.

What would you have decided, and what would you have measured first?

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 built the set. It took two and a half days, not three. A hundred and twenty real doubts came off last year's phones: 55 typed conceptual questions, 40 photographed numericals, 15 that were really a student asking why their own working was wrong, and 10 that were off-topic or abusive and should have been declined.

Before labelling the rest, the two teachers independently labelled the same twenty. They disagreed on seven. Not one of the seven was about physics. They were about what counted as "correct": one teacher accepted an answer with the right number and a wrong intermediate step, the other did not. They wrote the ruling down (the final answer and the method both have to be right for a numerical; for a conceptual doubt, the answer has to address the misconception the student actually has). On the next twenty they disagreed on two.

The result: 71% overall. On typed conceptual questions, 89%. On photographed numericals, 38%, mostly because the photographs were poor and the model solved a slightly different question from the one in the picture. Two answers out of 120 were confident wrong numericals with no hedge at all. That failed Mrs Iyer's gate on its own.

They did not launch it in the groups. They launched the drafting version: the tool answered typed conceptual doubts directly, tagged as AI-generated, and everything containing a photograph or a number went to the morning queue with the draft attached. The junior teachers' evening shift went from four hours to about ninety minutes. Rakesh kept the 120 items as a frozen set and the failed photographs went into a separate folder that became the first thing they tested when the model was upgraded three months later.

Rakesh had fifteen correct answers out of fifteen. Why did that not settle whether the tool was ready?

Fifteen passes puts only a loose upper bound on the failure rate: a system failing a quarter of the time would still produce fifteen clean passes fairly often. The sample was also not representative, since the questions came from the centre's own printed module rather than from what students actually send, and the reader was the person least able to spot a plausible wrong answer. A result needs a number on a fixed, representative set, next to a comparison.

The two teachers disagreed on seven of the first twenty labels, and none of the disagreements were about physics. What does that tell you, and what should happen next?

It tells you the criterion was not yet defined, not that the teachers were careless. Disagreement clustered on what counted as correct, so the fix was a written ruling in the labelling guide and a fresh twenty to check agreement had improved. Until two people can label the same items the same way, no model can be scored against the label and no judge could later automate it.

The overall pass rate was 71%, below the 85% threshold, yet the centre shipped something. Was that a violation of the pre-registered decision?

No. The decision written in advance was between answering students directly and drafting for teacher approval; the threshold governed the first option. The per-slice results showed conceptual doubts passed comfortably while photographed numericals did not, and the two confident wrong numericals failed the separately scored gate. Shipping the drafting mode, with direct answers only on the slice that cleared, is exactly what the three-line decision file was written to permit.

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 A team writes twelve test messages, runs them through a new address extractor, and all twelve come back right. What does that result allow them to conclude?

A. That the true failure rate is close to zero, because twelve consecutive passes would be very unlikely if the system were failing at any real rate.

B. Only that catastrophic failure is unlikely; a system failing one time in four would often still pass twelve straight.

C. That the common case is handled and only rare cases remain to be tested.

D. Nothing whatsoever, since twelve items is too few to support any inference.

2 The course insists that the ship-or-hold threshold be written down before any result is seen. What is the mechanism it is guarding against?

A. A threshold written after the run tends to land wherever the result did.

B. The p-value is only valid when the threshold is a fixed parameter of the test.

C. Writing it first lets the team stop labelling as soon as the threshold is reached, saving effort.

D. The metric cannot be chosen until the threshold is known, so the order is forced.

3 Refund requests are 1% of traffic but, after stratified sampling, 12% of the evaluation set. How should a headline pass rate be quoted?

A. Quote the set-wide pass rate exactly as it stands; stratifying changes which items are present in the set, not how each of them is scored.
B. Remove refund items until they are 1% of the set, then quote the plain average.
C. Report only the refund pass rate, since rare strata are where systems fail.
D. Weight each stratum's rate by its real traffic share, and say the result is less certain than its item count suggests.

4 Two labellers agree on 95 of 100 posts, but only 2% of posts are harmful, and Cohen's kappa comes out near zero. What is the right reading?

A. The labellers are unreliable and should be retrained before any more labelling is done, since 95% is not good enough for a safety class.
B. Raw agreement is the trustworthy figure here and kappa should be ignored for rare classes.
C. Chance agreement is already about 94%, so kappa has little room; report the marginals and the overlap on positives too.
D. The definition of harmful is broken and the labelling guide must be rewritten from scratch.

5 A team generates 500 Devanagari test inputs from templates and the system passes 96% of them. Which claim does that result support?

A. About 4% of production failures come from Devanagari input.
B. The system can handle Devanagari input of the shapes that were generated.
C. The headline pass rate will rise if the 500 items are added to the traffic set.
D. Devanagari-speaking users will see a 96% success rate.

6 After twenty rounds of prompt tuning, the development set scores 91% and the rarely-run test set scores 76%. What is the most likely explanation?

A. The test set is harder than the development set and should be re-sampled to match it.

B. Run-to-run variance at temperature zero comfortably explains a fifteen-point gap on sets this size.

C. Near-duplicates have leaked from the test set into the development set.

D. The prompt has been fitted to the development set's particular items rather than improved.

MODULE 2

Metrics that mean something

A number is only useful if it maps onto the decision you are making. This block works through the metrics for the shapes of output you will actually meet — a label, a probability, a quantity, a ranked list and prose — the curves that summarise a system across every threshold at once, and what each of them hides when you average it.

> BY THE END OF THIS MODULE YOU CAN
>
> Choose a metric that matches the decision at hand, set a threshold from a real confusion matrix or a precision-recall curve, say whether a model's 0.8 means anything about probability, and separate a retrieval failure from a generation failure in a system that does both

---

11 · 8 min

What to measure when the output is prose

> THE ONE THING TO KEEP
>
> Decompose quality into checks that are separately true or false; one blended score hides where it broke.

Stop scoring "quality"

The instinct is to rate each output 1 to 5 for quality. Resist it. A 5-point score means something different to each labeller, something different to the same labeller on a Friday, and nothing at all to the engineer who has to act on it. "We went from 3.4 to 3.6" points at no change you can make.

Split the thing you care about into checks that are separately true or false. Five binary checks beat one 5-point score, every time, for the same labelling effort.

Three families of check, best first

Extract and compare exactly. Most "open-ended" tasks have a right answer buried in the prose. A summary of an invoice contains an amount. A reply about a refund contains a policy window. A support answer names a product that either exists or does not. Pull that value out and compare it with `==`. This feels crude and it is the strongest signal you will get — it does not drift, it does not need a judge, and it is impossible to argue with.

Rubric checks, binary, written as questions. "Does it reply in the language the customer wrote in?" "Does it avoid promising a refund?" "Is it under 120 words?" "Does every link it gives resolve?" Each is yes or no, each is cheap to check, and when the score drops you know which one moved.

Behavioural checks — did the next thing work. Did the JSON parse. Did the SQL run without error. Did the tool call have valid arguments. Did the customer's next message say "thank you" rather than "no, I meant the other order". These cost nothing to collect and they are the closest thing to ground truth you will ever have.

```
def grade(item, output):
    return {
        "parses":        is_json(output),
        "amount_exact":  extract_amount(output) == item["ex-
pected"]["amount"],
        "currency_exact": extract_currency(output) == item["ex-
pected"]["currency"],
        "same_language": detect_lang(output) ==
detect_lang(item["input"]),
        "no_refund_promise": not PROMISE_RE.search(output),
    }
    # store the dict, per item. Never collapse to one number
here.
```

Keep every check separately for every item. Report the pass rate of each check and the pass rate within each tag. An overall number is for the meeting; the

per-check table is for the work.

The metrics that will waste your week

BLEU, ROUGE, and embedding-similarity-to-a-reference are widely used and mostly wrong for this. They were built for machine translation and for summarisation scored against reference corpora, where many valid outputs share vocabulary with the reference. Your task is not that. A support reply can share almost no words with your reference answer and be perfect; it can share most of them and be confidently wrong about the refund window.

When a similarity score moves, the honest reading is "the wording moved", not "the quality moved". Occasionally that is what you want — detecting that a prompt change made outputs drift stylistically is a real use. Just do not report it as accuracy.

A note on the ones you cannot check

Some things genuinely resist binary checks: tone, whether an explanation is at the right level, whether a piece of writing is any good. You have two honest options. Have humans rate them, in pairwise comparisons rather than absolute scores, on a small sample. Or use a model as a judge — which works, and misleads in specific ways that the next lesson is entirely about.

What you should not do is invent a number for them and put it on a dashboard, because a number on a dashboard gets optimised, and a fake number gets faithfully optimised into a worse product.

BEFORE YOU MOVE ON

A team measures its summarisation feature by cosine similarity between the generated summary and a human-written reference. After a prompt change the mean similarity rises from 0.71 to 0.79. What has been established?

A. That the new summaries use wording closer to the references; whether they are more accurate has not been tested.
B. That the summaries improved meaningfully — eight points on a zero-to-one scale is a large move.
C. Nothing, because embedding similarity between texts is essentially random noise.
D. A valid improvement that now needs a significance test across the 100 items before shipping.

12 · 9 min

Precision and recall, worked on real numbers

THE ONE THING TO KEEP

Precision, recall and accuracy come from the same four cells, and under class imbalance accuracy rewards a system that does nothing.

Four cells, and everything comes from them

Any yes/no classifier produces four outcomes on a labelled set. A content filter run over 1,000 posts, of which 60 are genuinely harmful:

- It flagged 80 posts.
- Of those 80, 48 really were harmful. These are **true positives**.
- The other 32 flagged posts were fine. **False positives**.
- 12 harmful posts went through unflagged. **False negatives**.
- 908 fine posts were correctly left alone. **True negatives**.

That is the confusion matrix. Every metric in this lesson is arithmetic on those four numbers, and you should be able to do all of it on paper.

A content filter over 1,000 posts, 60 of them harmful

	Flagged	Not flagged
Actually harmful	**48** true positives	**12** false negatives
Actually fine	**32** false positives	**908** true negatives

Precision 48/80 = 0.60, recall 48/60 = 0.80, accuracy 95.6%. A filter that flags nothing at all scores 94.0% accuracy from the bottom-right cell alone.

Precision — when it flags, how often is it right?

```
precision = TP / (TP + FP) = 48 / 80 = 0.60
```

Recall — of the things that should have been caught, how many were?

```
recall = TP / (TP + FN) = 48 / 60 = 0.80
```

F1 — the harmonic mean of the two, used when you want one number:

```
F1 = 2 * P * R / (P + R) = 2 * 0.6 * 0.8 / 1.4 = 0.686
```

Accuracy is the trap

Accuracy is `(TP + TN) / total` = `(48 + 908) / 1000` = **95.6%**. It sounds like a good system.

Now consider a system that flags nothing at all. It has zero true positives, zero false positives, and 940 true negatives. Its accuracy is **94.0%**.

A classifier that does literally nothing scores within 1.6 points of the working one. That is what accuracy does under class imbalance, and imbalance is the

normal case: fraud, spam, defects, safety violations, disease, churn. The rarer the thing you care about, the more accuracy measures your ability to say "no".

Whenever somebody reports accuracy on a rare event, ask for the base rate. If they do not know it, they do not have a result.

Which of the two you care about is a business question

Precision and recall trade against each other and nothing in the mathematics tells you where to sit. The product does.

- **Recall matters more** when a miss is the expensive failure and a false alarm is merely annoying: screening for a serious disease, catching child-safety violations, detecting a payment outage. You accept reviewing extra cases.
- **Precision matters more** when acting on a positive is costly or irreversible and misses are recoverable: auto-deleting a customer's post, auto-refunding money, sending an accusatory email. Here, a wrong action costs trust you cannot buy back.
- **Neither, alone, when a human is downstream.** If flagged items go to a review queue, the real constraint is the queue's capacity. Precision then translates directly into wasted reviewer minutes, and the sane metric is "harmful posts caught per reviewer-hour".

F1 hides this choice by averaging it away. Report precision and recall as a pair and quote F1 only as a summary, never as the thing you optimise without saying which side you would rather err on.

More than two classes

With several categories — say routing tickets to billing, technical, sales, other — you compute precision and recall per class, then average. How you average changes the story:

- **Macro average** treats every class equally. A class with 8 examples counts as much as one with 800. Use it when the rare classes matter.

- **Micro average** weights by frequency, so it is dominated by the big classes and equals accuracy in the single-label case.
- **Weighted average** sits in between.

A router can have a macro-F1 of 0.52 and a micro-F1 of 0.88 at the same time, because it handles the two common categories well and the three rare ones badly. Both numbers are true. Only one of them tells you the sales queue is broken.

Always print the per-class breakdown before any average. In scikit-learn that is one call:

```
from sklearn.metrics import classification_report, confusion_-
matrix
print(confusion_matrix(y_true, y_pred, labels=labels))
print(classification_report(y_true, y_pred, labels=labels,
digits=3))
```

Both are free and run on a laptop; if you have no Python, `pandas.crosstab` or even a pivot table in LibreOffice Calc gives you the same four cells.

Read the matrix, not just the metrics

The cells themselves tell you what to fix. Print the full grid and look at where the mass sits off the diagonal. "Technical" being misread as "billing" 40 times is one specific, fixable confusion — often two categories that a human would also struggle to separate, which is a taxonomy problem, not a model problem.

This is the fastest route from a disappointing number to an actual next action, and it takes about ten minutes. A summary metric never tells you which two classes to merge.

BEFORE YOU MOVE ON

A defect detector runs over 5,000 parts, 100 of which are genuinely defective. It flags 60 parts and 45 of those are truly defective. Which statement is correct?

A. With 98.8% accuracy the detector is performing well and is ready to run unattended.
B. Recall is 45%, so more than half the defective parts are shipping, even though precision is a respectable 75%.
C. Precision is 45% and recall is 75%, so the main cost is wasted inspection time.
D. F1 of 0.56 shows the detector is close to random and the approach should be abandoned.

13 · 8 min

When an item has several right answers

THE ONE THING TO KEEP

Exact-match scoring on multi-label tasks collapses under its own arithmetic, so score at the level the product acts on and audit predicted labels by hand, because the gold set is rarely complete.

One ticket, three labels

Real classification tasks are rarely one label per item. A support message says the payment failed, asks for a refund, and threatens to leave. A post is both political and abusive. A product belongs to three categories. The moment more than one label can be true at once, most of the arithmetic from the confusion-matrix lesson needs a decision before you can apply it.

Start with the strictest possible rule, **exact match**: the prediction is correct only if the predicted label set equals the true label set. It is easy to compute and it punishes you brutally, for a reason worth seeing in numbers. If items

carry five labels and each is predicted correctly 90% of the time, independently:

```
exact match = 0.90 ** 5 = 0.59
```

A system that is right about every individual label nine times in ten looks like a 59% system. Nothing is broken; the metric is multiplying. Report exact match only where the product genuinely needs the whole set right — populating a form that a human will not review, for instance.

The three usable alternatives

Per-label precision and recall. Treat each label as its own binary problem. This is the diagnostic view and it is where you should always start, because it shows that the failure is concentrated in two labels out of forty.

Sample-averaged F1. For each item, compute the overlap between the predicted and true sets, then average across items. For an item where you predicted `{billing, refund}` and the truth was `{billing, refund, churn-risk}`: precision 2/2, recall 2/3, F1 0.8. This answers "how good is a typical prediction", which is closest to what a user experiences.

Hamming loss. The fraction of individual label decisions that are wrong across the whole grid. It is stable and it is nearly useless on its own, because with forty labels and two true ones, always predicting nothing gives a Hamming loss under 5%.

```
from sklearn.metrics import classification_report, f1_score
print(classification_report(Y_true, Y_pred,
target_names=labels))   # per label
f1_score(Y_true, Y_pred, average="samples")
# per item
```

Thresholds are per label, not global

Multi-label models usually emit an independent score per label. A single global cutoff of 0.5 will be far too high for rare labels and too low for common

ones, because the model's score distribution differs per label with the training frequency.

Tune one threshold per label on the development set, maximising whatever that label needs — recall for the safety label, precision for the label that triggers a refund. It is a loop of ten lines and it routinely gains more than a model change. Record the thresholds with the model version; a threshold table that lives in somebody's notebook is a production incident waiting to happen.

Taxonomies: wrong, and wrong by how much

When labels form a hierarchy — `refund` above `refund-partial` and `refund-full` — flat scoring treats a `refund-partial` prediction on a `refund-full` item exactly like predicting `billing`. That is wrong in a way that matters, because the two errors have different consequences.

The fix is not a clever metric. It is to score at the level the product acts on:

- If the label routes the ticket to a queue and both children go to the same queue, evaluate at the parent level, where the system is right.
- If the label sets the refund amount, evaluate at the leaf level, where the system is wrong.

Compute both and report them as two lines: "parent-level accuracy 91%, leaf-level 68%". This says precisely what works and what does not, and it stops the argument about whether the model is good.

The gold set is probably incomplete

Here is the failure specific to multi-label work, and it catches good teams. Annotators tag the labels they notice. On a long message they tag two and miss the third. Your model predicts the third correctly, and your evaluation counts it as a false positive.

The consequence is systematic: measured precision is biased downwards, and the bias grows with how good the model is. A model that finds labels humans overlook scores worse.

The way out is a targeted audit. Take 50 predicted labels that the gold set does not contain and have a person judge each one — not "did the annotator write this", but "is this true of this item". You get a corrected precision and, usually, a list of annotation-guide holes. Run this before you conclude that precision is falling; roughly a third of the time the model is right and the file is out of date.

BEFORE YOU MOVE ON

A multi-label classifier is right about each individual label 92% of the time, and items carry four labels on average. Exact-match accuracy comes out near 72%. What should you conclude?

A. The model has a systematic bias on multi-label items and needs retraining with class weights.

B. Exact match multiplies per-label accuracy across labels, so 72% is what a strong per-label model looks like, and the useful views are per-label and per-item scores.

C. Exact match is invalid for multi-label problems and should never be reported.

D. Hamming loss would show the same 28% failure rate more clearly, so it should replace exact match.

14 · 9 min

What a curve says, and what its one number leaves out

THE ONE THING TO KEEP

AUC is the probability that a random positive outranks a random negative, so it measures ranking across thresholds you would never run — quote recall at a fixed precision, or precision at your real review budget, instead.

AUC is a statement about pairs

Everybody quotes ROC-AUC and few people can say what it means. It has an exact and useful interpretation: **pick one positive item and one negative item at random; AUC is the probability that the model gives the positive the higher score.**

You can compute it by counting. Suppose three positives score 0.9, 0.6, 0.4, and four negatives score 0.8, 0.5, 0.3, 0.1. There are $3 \times 4 = 12$ pairs. Count the pairs where the positive is higher:

- 0.9 beats all four.
- 0.6 beats 0.5, 0.3, 0.1 — three.
- 0.4 beats 0.3, 0.1 — two.

Nine of twelve, so AUC = 0.75. No curve required. This is the Wilcoxon–Mann–Whitney statistic, and knowing it is the same thing as AUC explains everything else in this lesson.

Three consequences that follow immediately

AUC ignores calibration. Pass every score through any function that preserves order — square them, take logs, add 0.4 — and every pair comparison is unchanged, so AUC is unchanged. A model with beautiful AUC can emit

scores that mean nothing as probabilities. That is a separate property, measured separately, in the next lesson.

AUC averages over thresholds you will never use. The curve includes the region where you flag 80% of all traffic. If nobody would ever run there, the model's skill in that region is being counted towards the number you are judging it by. Two models can tie at 0.86 while one is better where you actually operate and worse everywhere else.

AUC is insensitive to the positive rate. The false-positive-rate axis has all the negatives in its denominator, so with 4% positives an extra hundred false alarms moves the curve imperceptibly while quintupling your review queue. This is why average precision — the area under the precision–recall curve — is the honest summary for rare positives.

Curves cross, and the crossing is the decision

Plot both models on the same precision–recall axes. It is common for model A to hold higher precision in the high-recall region while model B is better when you only want the top few per cent. A single scalar cannot represent that, and whichever scalar you quote will pick a winner by accident.

So quote the number that matches your operating point:

- **Recall at fixed precision.** "At 90% precision, A catches 41% of cases and B catches 33%." This is the right framing when a false positive has a fixed unacceptable cost.
- **Precision at a fixed budget.** "Reviewing 200 items a day, A's queue is 62% real and B's is 51%." This is the right framing whenever a human queue exists, because the queue length is the real constraint.
- **False positive rate at a fixed recall.** Common in fraud and safety, where the recall target is set by policy.

```
from sklearn.metrics import precision_recall_curve
p, r, t = precision_recall_curve(y_true, scores)
i = (p >= 0.90).argmax()          # first point meeting the
precision target
print(f"recall {r[i]:.2f} at precision {p[i]:.2f}, threshold
{t[i]:.3f}")
```

Four lines, free, and it produces a sentence a product manager can act on rather than a number only you can interpret.

The uncertainty nobody prints

A curve drawn from 40 positives is mostly noise, and average precision computed on it swings several points between resamples. The mechanism is direct: precision at the top of the ranking is computed from a handful of items, so one item moving changes it a lot.

Bootstrap the whole curve — resample items with replacement, recompute the metric, repeat 1,000 times, take the 2.5th and 97.5th percentiles. That is the next module's tool, and this is the first place you need it. If the two models' intervals overlap heavily, you do not yet have a result, however clean the plot looks.

Plot it once, even if never again

Print the curve for your own system at least once. Three shapes are diagnostic and instantly readable:

- A curve hugging the top-left means a threshold exists that gives you both precision and recall. Take it.
- A curve that falls off a cliff at 20% recall means the model is confident about a small easy subset and guessing thereafter — often a sign that an easy pattern dominates the training data.
- A curve near the diagonal means no threshold will save you, and no amount of threshold tuning is worth doing until the model changes.

Ten minutes with `matplotlib` settles which of those three situations you are in, and each has a different next action.

BEFORE YOU MOVE ON

Two spam classifiers both score ROC-AUC 0.91 on the same 5,000-message set with 3% spam. Which follow-up tells you most about which to ship?

A. Compute F1 for both at the default 0.5 threshold and pick the higher one.

B. Compare precision at the volume your review queue can actually absorb, and plot both precision-recall curves to see where they cross.

C. Compare their accuracies, since AUC ignores the base rate and accuracy does not.

D. Prefer the model with better-calibrated probabilities, since equal AUC means equal ranking quality.

---

15 · 10 min

Moving the threshold is a business decision

THE ONE THING TO KEEP

The threshold is not a model parameter but a price you set on the two kinds of mistake, and a band with a human in the middle usually beats a single line.

The model gives a score; you choose the line

Most classifiers do not output yes or no. They output a number between 0 and 1, and somebody — usually by default, usually without noticing — decided that 0.5 means yes.

0.5 is not a principled choice. It is the midpoint of a range, and the range is arbitrary. The threshold is where you convert a score into an action, and it should be set by what the two kinds of mistake cost you.

Take the content filter from the previous lesson: 1,000 posts, 60 harmful. Sweep the threshold and the picture changes completely.

- At **0.3**: flags 210 posts, catches 57 of 60. Recall 95%, precision 27%. The review queue has 153 innocent posts in it.
- At **0.5**: flags 80, catches 48. Recall 80%, precision 60%.
- At **0.8**: flags 38, catches 34. Recall 57%, precision 89%. Twenty-six harmful posts go through.

Same model, same weights, three genuinely different products. Nobody retrained anything.

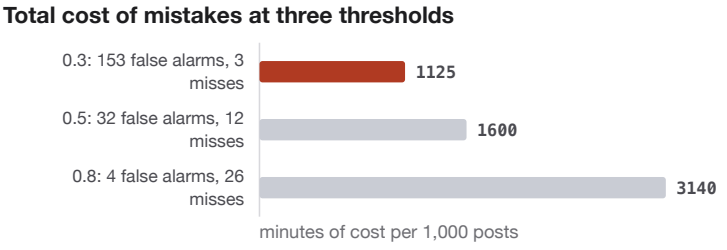
Put a number on each mistake

You can do better than arguing about it. Assign a cost to a false positive and a false negative, in any consistent unit, and pick the threshold that minimises total cost.

Say a false positive costs you five minutes of a moderator's time, and a false negative costs the equivalent of two hours of harm, complaints and trust. That is a ratio of 24 to 1.

```
cost = FP * 5 + FN * 120     (in minutes)

threshold 0.3:  153*5 + 3*120  =  765 + 360  = 1125
threshold 0.5:   32*5 + 12*120 =  160 + 1440 = 1600
threshold 0.8:    4*5 + 26*120 =   20 + 3120 = 3140
```

The low threshold wins by a wide margin, and now the argument is about the 24:1 ratio, which is the argument you should have been having all along. People who will not agree on a threshold will often agree on a ratio, and the ratio is the honest form of the disagreement.



At five minutes per false alarm and two hours per miss, the low threshold wins by a wide margin. Change the 24:1 ratio and the winner moves, which is the argument worth having.

The numbers are made up. Making them up explicitly, in a document, is still much better than leaving them implicit in a default.

PR curves and the ROC trap

Sweeping the threshold and plotting the results gives you a curve, and there are two conventions.

The **ROC curve** plots true positive rate against false positive rate, and its area (AUC) is the number most commonly quoted. Under heavy imbalance it is misleading, because the false positive rate has 940 true negatives in its denominator. Going from 32 to 153 false positives moves that rate from 3.4% to 16% — the curve barely notices, while your review queue has grown fivefold.

The **precision–recall curve** puts precision on one axis, and precision has the flagged count in its denominator, so it feels every one of those false positives. For rare positives, read the PR curve. Report average precision rather than ROC-AUC.

Both are one line in scikit-learn (`precision_recall_curve`, `roc_curve`), free, and worth plotting once even if you never plot it again — seeing your own curve's shape tells you whether there is a threshold that gets you both, or whether the model is simply not separating the classes and no threshold will save you.

Scores are not probabilities

A score of 0.9 does not mean "right 90% of the time" unless the model has been calibrated. Check it directly: bucket predictions into ten bins by score,

and in each bin compare the mean predicted score to the observed fraction of positives. If the 0.9 bucket is right 62% of the time, your scores are inflated, which is normal for neural networks.

Fixing it is cheap — Platt scaling (a logistic regression on the scores) or isotonic regression, both in `sklearn.calibration`, both needing only a held-out set with labels. Calibrate before you compute expected cost, because the cost arithmetic above assumes the score means something.

For a large language model asked to rate its own confidence, none of this applies and calibration is worse than poor. Self-reported confidence clusters near 0.9 regardless of correctness. If you need a usable score from an LLM classifier, take the token log-probabilities of the label tokens where the provider exposes them, and calibrate those. Treat a number the model has typed out in prose as text, not as a probability.

Two thresholds are often better than one

The most useful shape in practice is not one line but two. Below 0.3, act automatically as negative. Above 0.85, act automatically as positive. Between, send it to a human.

This lets you tune the middle band to your actual review capacity: if the queue can absorb 200 items a day, widen or narrow the band until it does. You get high precision on both automatic paths and you spend your scarce human attention exactly where the model is unsure. It is also the design that degrades gracefully, because when volume spikes the band can be narrowed rather than the whole system switched off.

BEFORE YOU MOVE ON

A team reports ROC-AUC of 0.94 for a fraud model where 0.8% of transactions are fraudulent, and proposes shipping at the default 0.5 threshold. What is the most substantive objection?

A. AUC cannot be compared across datasets, so the 0.94 is meaningless without a baseline.

B. The model has not been retrained on recent data, so the score distribution is probably stale.

C. 0.94 is not high enough for a production fraud system; anything under 0.99 will generate too many alerts.

D. At a 0.8% base rate, ROC-AUC hides the false-positive burden and 0.5 is an arbitrary operating point; they should read the precision–recall curve and pick a threshold from the cost of each mistake.

16 · 9 min

Making a 0.8 mean eighty per cent

THE ONE THING TO KEEP

Calibration is measured with a reliability diagram and summarised by expected calibration error or log loss; fitting it changes no rankings at all, which is exactly why it is cheap and why it cannot rescue a weak model.

The moment it matters

Two systems score identically on ranking. One of them lets you write this line:

```
expected cost = p * cost_of_acting_wrongly + (1 - p) *
cost_of_not_acting
```

The other does not, because its `p` is a number-shaped noise. Any decision built on expected cost, any threshold derived from money, any triage that says "route everything above 70% risk to a human" depends on the score meaning what it says. That property is calibration, and it is measured, not assumed.

Measure it with bins

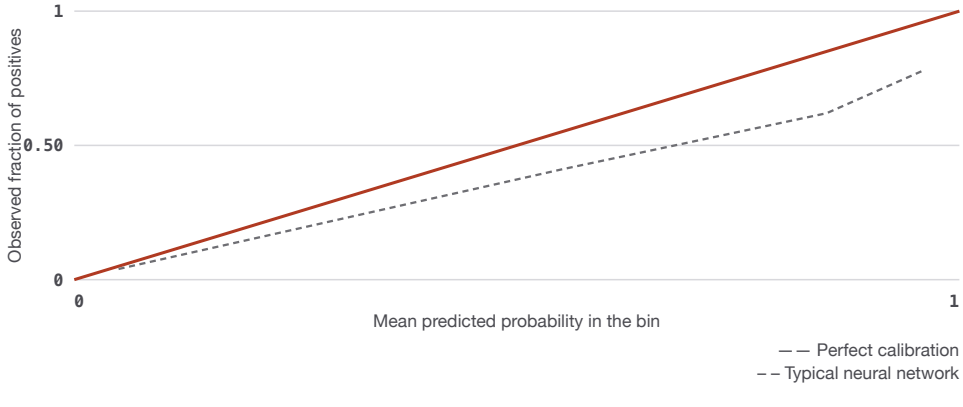
Take a held-out labelled set, sort by predicted score, cut into ten bins, and compare each bin's mean predicted score with the observed fraction of positives in it.

A typical neural network gives you something like this:

- bin 0.0–0.1: predicted 0.05, observed 0.04 — near enough
- bin 0.5–0.6: predicted 0.55, observed 0.41 — over-confident
- bin 0.8–0.9: predicted 0.85, observed 0.62 — badly over-confident
- bin 0.9–1.0: predicted 0.96, observed 0.78 — badly over-confident

Plotted, that is the reliability diagram, and modern classifiers usually bulge below the diagonal: they are more sure than they should be, because the training objective rewards pushing probabilities to the extremes.

Reliability diagram for a typical classifier



Every point below the diagonal is over-confidence: the 0.85 bin is right 62% of the time. Platt or isotonic scaling pulls the curve onto the line without changing a single ranking.

Expected calibration error summarises it as the count-weighted mean of the gaps.

```
import numpy as np
def ece(p, y, bins=10):
    edges = np.linspace(0, 1, bins + 1)
    idx = np.clip(np.digitize(p, edges) - 1, 0, bins - 1)
    return sum(np.mean(idx == b) * abs(p[idx == b].mean() -
y[idx == b].mean())
               for b in range(bins) if (idx == b).any())
```

An ECE of 0.03 means your probabilities are off by three points on average. An ECE of 0.15 means the word "probability" is doing no work.

Two proper scores complement it. **Brier score** is the mean squared error of the probabilities. **Log loss** is the mean of `-ln(p_assigned_to_the_truth)` and punishes confident mistakes savagely: being wrong at 0.9 costs `-ln(0.1) = 2.30`, being wrong at 0.6 costs `-ln(0.4) = 0.92`. If your product suffers most from confident errors, log loss is the metric that agrees with you.

Fixing it is cheap

Fit a small monotone function from raw score to calibrated probability on a held-out slice.

- **Platt scaling** fits a one-parameter logistic curve. It needs only a few hundred labelled points and is hard to overfit. It assumes the miscalibration has a sigmoid shape, which it usually does.
- **Isotonic regression** fits any non-decreasing step function. It corrects shapes Platt cannot, and it needs a few thousand points or it memorises the calibration set.
- **Temperature scaling** is Platt's single-parameter cousin applied to logits, standard for neural networks: divide the logits by one learned number.

```
from sklearn.calibration import CalibratedClassifierCV
calibrated = CalibratedClassifierCV(model, method="sigmoid",
cv="prefit").fit(X_cal, y_cal)
```

Here is the property that makes this safe and also limits it: all three are monotone, so **they change no rankings at all**. AUC, average precision, and

every recall-at-precision figure stay exactly where they were. Calibration makes the number usable; it cannot make a weak model strong, and a perfectly calibrated model that always outputs the base rate is perfectly calibrated and perfectly useless.

Two ways calibration breaks after you fix it

It is not uniform across groups. A model calibrated overall can be overconfident for one language and under-confident for another, and the two errors cancel in the average. Compute ECE per slice, not once. This is the same arithmetic that makes group fairness hard, and it appears again later in the course.

It moves when the base rate moves. Calibrate at 4% prevalence, deploy where prevalence is 12%, and every probability is wrong. If only the prevalence has shifted, correct the odds directly rather than refitting:

```
new_odds = old_odds * (0.12 / 0.88) / (0.04 / 0.96)
```

Prevalence drifts with season, with a marketing campaign, with a new market. Re-check calibration on fresh labelled data on a schedule, the same way you re-check the model.

The special case of language models

A model asked "how confident are you, 0 to 100" produces text, not a probability, and that text clusters around 80 to 95 almost regardless of correctness — the phrasing of confident answers is what the training data rewarded. Where the provider exposes token log-probabilities for the answer tokens, those can be calibrated with the tools above and behave sensibly. Where it does not, treat any stated confidence as a style choice, and get your uncertainty from agreement across several samples instead.

BEFORE YOU MOVE ON

A team applies isotonic regression and finds that expected calibration error falls from 0.14 to 0.02 while ROC-AUC is unchanged to three decimal places. What is the correct interpretation?

A. The calibration fit failed, since a genuine improvement in the probabilities should also improve ranking quality.

B. AUC was already at ceiling for this model, so no further ranking gain was available.

C. The calibration worked as designed: monotone rescaling preserves ordering, so probabilities became usable for expected-cost decisions while the model's ability to separate classes stayed identical.

D. Isotonic regression overfitted the calibration set, which is why the improvement did not carry over into AUC.

17 · 9 min

When the output is a quantity

THE ONE THING TO KEEP

Choose between MAE and RMSE by deciding whether one large error is worse than several small ones, avoid MAPE because its asymmetry biases model selection towards under-forecasting, and score every forecast against the naive baseline.

The same question in a different costume

Plenty of AI features output a number: predicted delivery time, forecast demand, an estimated price, a risk score in rupees, a count extracted from a document. The evaluation question is unchanged — what decision does this serve — but the metrics differ, and the differences are not cosmetic.

Start with five errors: 1, 1, 1, 1, 10 units.

```
MAE  = (1+1+1+1+10) / 5              = 2.8
RMSE = sqrt((1+1+1+1+100) / 5)       = sqrt(20.8) = 4.6
```

Mean absolute error is the average miss, in the units of the thing. **Root mean squared error** squares first, so the single large error dominates. That is the whole choice, and it is a business question rather than a statistical one:

- If ten small delays cost the same as one big delay, use MAE.
- If one delivery three hours late costs more than ten deliveries eighteen minutes late — because it triggers a refund, a call, a lost customer — use RMSE, which will steer the model away from large misses.

Quote MAE alongside anything else, because it is the only one a non-specialist reads correctly: "on average we are 2.8 minutes out" is a true sentence about MAE and a false one about RMSE.

Why MAPE keeps getting chosen and keeps causing trouble

Mean absolute percentage error is popular because it is unit-free and comparable across products. It has two mechanical defects that change decisions.

It explodes near zero. Actual 2, predicted 4: the absolute error is 2 and the percentage error is 100%. Actual 100, predicted 98: the same absolute error is 2%. On any series with small values — a slow-moving product, a quiet night — MAPE is dominated by items nobody cares about.

It is asymmetric. Over-forecasting can never exceed 100% error when the actual is fixed, while under-forecasting is unbounded. So a model trained or selected on MAPE learns to forecast low. This is not a subtlety; it shows up as systematic under-stocking in real inventory systems.

Use **WAPE** instead, which is the total absolute error divided by the total actual volume:

```
WAPE = sum(|actual − predicted|) / sum(actual)
```

It is unit-free, it does not explode, and it weights by size, which is usually what the business means by "how far off were we".

Compare against the naive forecast, always

For anything with a time dimension, the baseline is not the mean. It is **last value** ("tomorrow equals today") or **seasonal naive** ("this Tuesday equals last Tuesday"). Both are free and both are hard to beat.

MASE makes the comparison a single number: your MAE divided by the naive method's MAE on the same data.

- MASE 0.8 — you are 20% better than repeating last week.
- MASE 1.0 — you have built a machine that repeats last week.
- MASE above 1.0 — you would do better deleting the model.

R^2 carries the same idea for non-time-series regression: it compares your squared error against always predicting the mean, so a negative R^2 means you are worse than a constant. Both numbers exist to stop a team celebrating a forecast that a spreadsheet formula beats.

A point estimate is often the wrong output

Many decisions need a range, not a number: how much stock to hold, how much buffer to promise, whether to escalate. Then the model should emit an interval, and the evaluation is a coverage check.

Take the stated 90% interval and count how often the truth fell inside it. If it is 71%, your intervals are too narrow and every plan built on them is under-buffered. If it is 99%, they are too wide to be useful. This is exactly calibration from the previous lesson, applied to quantities, and quantile loss (also called pinball loss) is the training-time version of the same idea.

Report error by size, not once

Errors are almost never uniform. A 5% error on a ₹200 order and a 5% error on a ₹2,00,000 order are different businesses. Bucket by magnitude and print MAE and WAPE for each bucket. The usual finding is that the headline is car-

ried by the many small items while the money sits in the few large ones — and that the model is worst exactly there, because large values are rare in training data.

Everything here runs in `sklearn.metrics`, in `statsmodels`, or in a spreadsheet with four columns. None of it needs a GPU, and the naive baseline needs no model at all.

BEFORE YOU MOVE ON

A demand forecast is selected by lowest MAPE. Six months later the warehouse is persistently under-stocked. What is the most likely mechanism?

A. MAPE weights all products equally, so the model optimised for the many small products and neglected the large ones.

B. MAPE penalises under-forecasting less than over-forecasting for a given absolute error, because over-forecast error is bounded relative to the actual while under-forecast error is not, so selection favoured models that forecast low.

C. MAPE is undefined when the actual is zero, so those days were dropped and the model never learned about stock-outs.

D. RMSE should have been used, since squaring errors is the only way to penalise large misses.

18 · 9 min

Scoring a ranked list

THE ONE THING TO KEEP

For a retrieval-augmented system, measure recall at the number of passages you actually pass to the model, because the generator cannot recover an answer that was never retrieved.

When the output is an ordered list

Search, recommendation and the retrieval half of any RAG system all produce the same shape of output: a ranked list, of which the user or the model downstream sees the top few. Precision and recall need adapting, because position matters — the right document at rank 1 and the right document at rank 9 are not the same result.

Set up the evaluation the same way as everything else in this course. Fifty to a hundred real questions, and for each one, the identifiers of the documents that genuinely contain the answer. Building that mapping is the work; once you have it, every metric below is arithmetic.

A note on collecting it: do not write the questions by reading your documents. You will write questions shaped like the text, and the system will look better than it is. Take real questions — from search logs, support tickets, the sales inbox — and then find which document answers each one.

The four metrics, worked

A query with two relevant documents, `D3` and `D7`. The system returns:

```
rank 1: D9   (not relevant)
rank 2: D3   (relevant)
rank 3: D5   (not relevant)
rank 4: D2   (not relevant)
rank 5: D7   (relevant)
```

Recall@k — what fraction of the relevant documents appear in the top k?

```
recall@3 = 1/2 = 0.50
recall@5 = 2/2 = 1.00
```

Precision@k — what fraction of the top k are relevant?

```
precision@3 = 1/3 = 0.33
precision@5 = 2/5 = 0.40
```

MRR — reciprocal rank of the *first* relevant result, averaged across queries. Here the first relevant hit is at rank 2, so `1/2 = 0.5`. Use MRR when the user needs one good answer and will stop reading once they find it.

nDCG@k — discounted cumulative gain, normalised. It gives credit that falls off with position (`1/log2(rank+1)`) and, unlike the others, handles graded relevance: a document can be perfect, partly useful or useless rather than only relevant or not. Use it when relevance is genuinely a matter of degree, and be aware that it is the metric people quote most and inspect least.

For RAG, recall@k is the one that matters

Here is the practical instruction. If you are retrieving passages to put into a model's context, measure **recall@k where k is the number of passages you actually pass in**. Nothing else comes close in importance.

The reason is a hard constraint. If the answer is not in the retrieved context, no prompt, no model upgrade and no clever instruction can produce a correct grounded answer. The generation step cannot recover information that was never handed to it. It can only decline, or invent.

Teams routinely spend three weeks rewriting prompts to fix a system whose retrieval recall@5 is 0.55. Measure it first. It takes an afternoon, and about half the time it ends the investigation on the spot.

Ordering *within* the retrieved set matters less than people expect for RAG, because the model reads all of it — though not evenly: content in the middle of a long context is used less reliably than content at either end. Keep k modest and put the strongest passage first.

Where retrieval quietly fails

When recall@5 is poor, the cause is usually one of four things, and they are distinguishable by reading twenty failed queries:

- **Chunking.** The answer spans a boundary, so no single chunk contains it. Overlapping windows, or chunking on document structure rather than a fixed character count, fixes more retrieval problems than any embedding-model change.
- **Vocabulary mismatch.** The user says "can't log in", the document says "authentication failure". Dense embeddings help here and are exactly why they exist — but they are weak in the opposite case.
- **Exact tokens.** Part numbers, error codes, names, dates. Embeddings blur these; keyword search does not miss them. Hybrid retrieval — BM25 and dense vectors, results merged with reciprocal rank fusion — beats either alone on most real corpora, and BM25 is free in `rank_bm25`, SQLite FTS5 or Postgres full-text search.
- **The document does not exist.** Roughly one failure in five, in my experience of looking, is a question your corpus genuinely cannot answer. That is a content problem, and no retrieval work will fix it. It also means your system needs a correct way to say so, which is the subject of the next lesson.

Tools

`ranx` and `pytrec_eval` compute all four metrics from a run file and are free and open source. BEIR is a standard suite of retrieval benchmarks, useful for

choosing an embedding model to start from, useless as a substitute for measuring on your own corpus. Honestly, twenty lines of Python over a dictionary of `{query_id: [relevant_doc_ids]}` will do everything in this lesson, and writing it yourself means you know what the numbers mean.

BEFORE YOU MOVE ON

A RAG assistant answers 58% of questions correctly. The team is comparing three prompt rewrites to improve it. What should they measure first instead?

A. Recall@k at the number of passages fed to the model, since a correct answer is impossible when the supporting passage was never retrieved.

B. nDCG@10 over the retriever, since it captures ranking quality better than any single-cutoff metric.

C. Precision@5, since irrelevant passages in the context distract the model and lower answer quality.

D. Judge agreement on the 58% figure, since the answer-correctness labels may themselves be unreliable.

---

19 · 10 min

Three questions, not one, for a retrieval system

THE ONE THING TO KEEP

Measure retrieval, groundedness and relevance separately, and always include questions your corpus cannot answer so you find out whether the system says so.

One number cannot diagnose a two-stage pipeline

A retrieval-augmented system does two separable things: it finds passages, then it writes an answer from them. "Answer correctness" fuses the two, so a

fall from 71% to 62% tells you a great deal about your morning and nothing about where to look.

Split it into three measurements, each with its own number:

1. **Retrieval.** Was the supporting passage in the context? Recall@k, from the previous lesson.
2. **Groundedness.** Is every factual claim in the answer supported by the passages that were supplied? Also called faithfulness.
3. **Relevance.** Does the answer address what was actually asked?

The combinations are diagnostic, and each points somewhere different:

- Retrieval fails, answer wrong: fix chunking and search. Nothing else will help.
- Retrieval succeeds, groundedness low: the model is adding material. Tighten the prompt, lower temperature, require quotation.
- Retrieval succeeds, groundedness high, answer still unhelpful: the passage is in the corpus but does not really answer the question. A content problem.
- Retrieval fails and the answer is confidently wrong: the worst quadrant, and the one nobody measures. Cover it below.

Measuring groundedness

Groundedness is checkable mechanically, which makes it one of the few LLM qualities you can evaluate cheaply and reliably.

The method: break the answer into atomic claims, then check each claim against the supplied context on its own.

```
Step 1 — decompose
Input: the answer text.
Output: a JSON list of standalone factual claims,
        each understandable without the others.

Step 2 — verify, one call per claim
CONTEXT: {retrieved passages}
CLAIM:   {one claim}

Quote the span of CONTEXT that supports the claim, or write
NONE. Then output {"quote": "...", "supported": true|false}.

groundedness = supported_claims / total_claims
```

One claim per call matters. Ask a model to check six claims at once and its errors correlate: it decides the answer "looks right" and waves all six through. Ask separately and you get six independent judgements, which is the whole point.

Requiring a quoted span, rather than a yes/no, is the other half. A judge that must point at evidence is markedly harder to fool, and you can verify the quote is a genuine substring of the context in one line of code — a free, deterministic check on your judge's honesty.

The refusal case nobody tests

Here is the most common gap in RAG evaluation, and it is easy to close.

Every set is built from questions the corpus can answer. So the system is never asked something the documents do not cover, and nobody finds out what it does then. In production it is asked constantly, and what it does is write a fluent, plausible, entirely invented answer.

Add 20 items to the set where the correct output is a refusal:

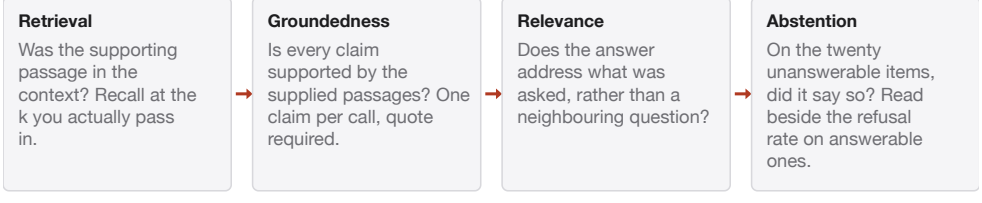
- questions about a product you do not sell
- questions about a period your documents do not cover ("what was the 2019 policy?" when your archive starts in 2022)

- questions with a false premise baked in ("why was the Mumbai office closed?" when it was not)
- questions that need information the corpus deliberately excludes, such as an individual's salary

Score these separately as **abstention rate**: the fraction where the system said it did not know rather than answering. Report it alongside accuracy, always. A system at 88% answer accuracy and 15% abstention on unanswerable questions is more dangerous than one at 79% and 90%, because the first one lies with a citation attached.

And track the mirror failure too — refusing questions the corpus *can* answer. Push abstention up carelessly and you get a system that says "I don't know" to everything, which scores beautifully on groundedness and is useless. The two numbers only mean something together.

Four numbers for a retrieval-augmented system



One 'answer correctness' figure fuses all four. Split apart, each failure points at a different fix: chunking and search, the prompt, the corpus, or the refusal behaviour.

Citations must be checked, not trusted

If your system shows sources, verify them programmatically on every eval run:

- the cited identifier exists in the corpus
- the cited document was actually in the retrieved context for that query
- the claim near the citation is supported by *that* document, not merely by some other retrieved one

The second and third checks catch a specific and common failure: the model attaches a plausible citation to a sentence it wrote from its own parameters.

Users trust a citation more than they trust prose, so a wrong citation costs more than a wrong sentence.

Tools, and what to build yourself

RAGAS (Apache 2.0) implements faithfulness, answer relevance and context precision out of the box. **TruLens** and **DeepEval** cover similar ground; **Inspect**, from the UK AI Safety Institute, is MIT-licensed and better suited if you want to write your own scorers. All four are free.

Use one to get moving in an afternoon. But build the decomposition-and-quote checker yourself at some point, because the packaged faithfulness scores are themselves LLM judges you have not calibrated, and module three is about to explain why that matters.

BEFORE YOU MOVE ON

A support RAG system scores 91% on answer accuracy across a 120-question eval set. Which addition would most change what the team knows?

A. Increasing the set to 400 questions, to tighten the confidence interval around the 91%.

B. Adding a second judge model and reporting the average of the two accuracy scores.

C. Adding 20 questions the corpus cannot answer, and scoring separately how often the system says so rather than inventing an answer.

D. Recording latency and cost per question alongside accuracy, so quality is not optimised in isolation.

20 · 8 min

Public benchmarks, and what they are for

THE ONE THING TO KEEP

Public benchmarks are a shortlist of models worth testing, never a decision, because contamination, format and their distance from your task all sit between the score and your product.

The numbers on the model card

Every model launch comes with a table: MMLU, GSM8K, HumanEval, MATH, SWE-bench, GPQA, and a position on a leaderboard. It is worth knowing what these actually measure, because they get quoted in decisions they cannot support.

- **MMLU** — about 16,000 multiple-choice questions across 57 school and university subjects. Four options, one correct. It measures recall of academic knowledge in a format nothing in your product resembles.
- **GSM8K** — 8,500 grade-school arithmetic word problems. Largely saturated: frontier models score above 95%, so differences are noise plus contamination.
- **HumanEval** — 164 small Python functions, each with unit tests. A model writes the body, the tests run. Real and verifiable, and 164 items is a very small set with a wide interval.
- **SWE-bench** — real GitHub issues from real repositories, scored by whether the patch makes the project's tests pass. Much closer to actual work, and much harder to game.
- **LMArena** — humans compare two anonymous responses and pick one, aggregated into an Elo rating.

The variation in construction matters. A benchmark scored by running tests is a different kind of evidence from one scored by matching a letter.

Contamination

These sets are on the public web. Models are trained on the public web. A model may have seen the questions and the answers during training, and no one outside the lab can verify that it did not.

The signs are recognisable once you know them: a suspiciously high score on an old benchmark alongside ordinary performance on a freshly written version of the same task; large sensitivity to trivial rewording that should not change difficulty; scores that fall sharply on a benchmark released after the model's training cutoff. Some benchmark authors embed *canary strings* — unique identifiers that should never appear in training data, so contamination can be detected afterwards. Almost nobody checks.

The practical consequence is not "benchmarks are fake". It is that a benchmark's absolute number means less than the difference between two models on it, and a two-point gap on MMLU means nothing at all.

The leaderboard has its own biases

Arena-style rankings are more robust than static benchmarks because the questions keep changing, but they measure a specific thing: which answer a volunteer prefers, after reading both, in a chat window, usually within thirty seconds.

That rewards formatting. Answers with headings, bullets and a summary line win against denser correct prose. It rewards length, up to a point. It rewards a confident tone, which is exactly the property you do not want in a system that must sometimes say it does not know. And the prompts come from people who volunteer to test chatbots, which is not your user base.

Style-controlled variants of these rankings exist and shuffle the order noticeably, which is itself the evidence that formatting was doing real work in the original.

What benchmarks are legitimately for

They are a **shortlist**, and that is a genuine use.

You cannot try forty models. Public scores, price and context window narrow it to three or four worth an afternoon each. A model that is far down on every coding benchmark is unlikely to surprise you on your coding task. A model with strong multilingual scores is a better first guess for Hindi support than one without.

What they cannot do is decide. The gap between benchmark rank and your task is enormous: your inputs are messier, your criteria are yours, your prompt does most of the work, and your latency and cost limits may exclude the top model anyway. Cases where the second- or third-ranked model wins on the actual task are common, usually for a boring reason — it follows format instructions more reliably, or it refuses less often on legitimate content in your domain.

The honest procedure

1. Shortlist three models from public scores, price and context length. Fifteen minutes.
2. Run your own hundred items against all three, same prompt, same day. An afternoon.
3. Compare on your metric, with intervals, plus cost and p95 latency.
4. Pin the winner by exact version string and record why. That record is what saves you when the model is deprecated in eight months.

Step two is the whole course. Steps one and four are why this lesson exists.

One last thing worth internalising: when a vendor claims a benchmark lead, the interesting question is not whether the number is true. It usually is. The question is whether that benchmark's task resembles yours, and for most products the honest answer is that it does not resemble it much at all.

BEFORE YOU MOVE ON

Model B beats Model A by 2.1 points on MMLU and sits three places higher on a public leaderboard. Your team builds an invoice-extraction feature. What follows?

A. Model B is the better general model, so it should be the default and A kept only as a fallback.

B. Both are worth an afternoon on your own hundred invoices; the benchmark gap is too small and too distant from extraction to decide anything.

C. Leaderboard position is the stronger signal because human preference is closer to real use than a multiple-choice test.

D. Neither number is usable because both benchmarks are contaminated, so public evaluation should be ignored entirely.

CASE STUDY · MODULE 2

The nurse who can read 250 messages a day

A district hospital in Nashik piloting a classifier that flags WhatsApp symptom messages for same-day attention

The hospital's outpatient department had started taking symptom messages on WhatsApp during the pandemic and never stopped. By 2026 it received about 2,000 messages a week: a photo of a rash, "my father has had fever for three days", a question about a prescription, a request to change an appointment. One nurse, Sunita, triaged them between her other duties and could realistically read about 250 in a working day.

A vendor offered a classifier. Each message got a score between 0 and 1 for "needs same-day clinical attention", and the vendor's deck said the model had an AUC of 0.91 on their evaluation set and an accuracy of 95%. The medical superintendent, Dr Pawar, was interested, and the hospital's one IT person, Prakash, was asked to make it work.

Prakash did the thing this course spends a module on. He took 500 messages from the previous two months, had Sunita and a junior doctor label each one as urgent or not, and found that 6% were genuinely urgent. Then he ran the vendor's model over them and swept the threshold.

At the default of 0.5, the model flagged 90 of the 500. Of those, 21 were really urgent, so precision was 23%. It missed 9 of the 30 urgent messages, so recall was 70%. Scaled up to a week's traffic of 2,000, that is 360 flagged messages, which Sunita could read, and roughly 36 urgent messages missed every week.

At 0.3, the model flagged 150 of the 500 and caught 28 of the 30 urgent ones. Recall 93%. But 150 of 500 is 30% of traffic, which is 600 flagged messages a week for a nurse who can read 250. The queue would be two and a half days behind by Wednesday, and an urgent message at the back of a queue is a missed message with extra steps.

At 0.8, the model flagged 24 and caught 16. Precision 67%, recall 53%. Almost half the urgent messages would go unflagged.

The vendor's accuracy figure was true and useless. A model that flagged nothing would score 94% accuracy on this data, because 94% of messages are not urgent.

Dr Pawar wanted the highest recall available. A missed urgent message was, in his words, the only failure that could end up in a newspaper. Sunita pointed out that the highest recall available would bury her, and that a flag she could not get to was worth nothing. Prakash had a third concern that neither of them had raised: the model's scores did not mean what they said. When he bucketed the 500 messages by score, the ones scored between 0.8 and 0.9 were urgent about half the time, not 85% of the time. The vendor's threshold advice, which assumed the score was a probability, was built on sand.

The decision had two parts. Where to put the line, given that every position traded a missed patient against a queue the nurse could not clear. And whether to trust a single line at all.

There was a budget question underneath it. The hospital could not hire a second nurse for this. It could, possibly, get a junior doctor to spend forty-five minutes a day on the queue. Forty-five minutes was perhaps another 80 messages.

How would you set the threshold, and what would you tell Dr Pawar about the 95% accuracy in the vendor's deck?

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 did not choose a line. They chose a band. Messages scoring above 0.85 triggered an automatic call-back from the desk, without waiting for Sunita. Messages below 0.15 got an automatic reply with the appointment link and the emergency number, and were still visible in a low-priority list. Everything in between went to Sunita's queue.

Prakash tuned the band edges on the 500 labelled messages until the middle came to about 220 messages a week at last month's volume, which was under Sunita's capacity with room for a bad week. At those settings the combined system caught 27 of the 30 urgent messages in the labelled set: 24 through the middle band and 3 through the automatic call-back. Two of the three misses were messages that did not look urgent to either human labeller until the second reading, which is an argument for a better labelling guide rather than a better model.

He calibrated the scores before setting the band, with a logistic fit on the 500 labels, so that the 0.85 edge meant something close to what it said. And he wrote the cost ratio down: a false alarm cost about two minutes of Sunita's time; a miss they valued at four hours. The number was made up, but making it up in a document meant that when Dr Pawar later wanted to widen the band, the argument was about the ratio and not about feelings.

The 95% accuracy figure went into a one-page note to the superintendent with the base rate written next to it, and the phrase "a model that flags nothing scores 94% on our data" underlined. The vendor's model stayed. The vendor's threshold advice did not.

The vendor reported 95% accuracy. Why was Prakash right to treat that as almost no information?

Only 6% of messages were urgent, so a model that flagged nothing would score 94% accuracy on the hospital's own data. Under class imbalance accuracy mostly measures the ability to say no. The numbers that describe the decision are precision and recall at a specific threshold, read together with the base rate.

Recall of 93% at a threshold of 0.3 sounds like the safest choice for patients. Why was it not?

At 0.3 the model flagged 30% of traffic, about 600 messages a week for a nurse who could read 250. A flag nobody reaches is a miss with a delay attached. The real constraint was the review queue's capacity, so the honest metric was urgent messages caught per nurse-hour, and a band with automatic handling at both ends spent Sunita's attention where the model was genuinely unsure.

Prakash calibrated the scores before setting the band. What would have gone wrong if he had skipped that step?

The band edges were chosen as if 0.85 meant an 85% chance of urgency, but the raw scores in that bucket were urgent only about half the time. Uncalibrated scores make any threshold reasoning built on expected cost wrong, even though calibration changes no rankings. A one-parameter logistic fit on a few hundred labelled points made the band edges mean what the team thought they meant.

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 A content filter flags 80 of 1,000 posts. Of the flagged posts, 48 are truly harmful, and 60 posts in total are harmful. What are precision and recall?

A. Precision 0.80 and recall 0.60.
B. Precision 0.48 and recall 0.80.
C. Precision 0.60, recall 0.80.
D. Precision 0.60 and recall 0.48.

2 On that same set, a filter that flags nothing at all scores 94% accuracy. Why does accuracy reward doing nothing?

A. Because accuracy counts true negatives, and 94% of the posts are negatives.
B. Because precision is undefined when nothing is flagged, so accuracy falls back to the base rate by convention.
C. Because F1 and accuracy coincide whenever the classes are imbalanced.
D. Because a filter that flags nothing is effectively at threshold 1.0, which maximises precision.

3 A moderation queue can absorb about 200 items a day. Which threshold design does the course recommend?

A. Pick the single threshold that maximises F1, since that balances precision and recall, and send everything above it to the queue.

B. Two thresholds: act automatically at both ends and send the band between them to the queue, tuned so it fills to 200.

C. Lower the threshold until recall exceeds 95%, then hire reviewers to match the resulting queue.

D. Keep the threshold at 0.5, because that is the point at which the score is calibrated.

4 A classifier has ROC-AUC 0.86 and an expected calibration error of 0.15. After Platt scaling on a held-out slice, what changes?

A. AUC rises, because the probabilities now agree with the outcomes.

B. Both fall, because calibration trades ranking quality for honest probabilities.

C. ECE falls and AUC is unchanged: the fit is monotone and reorders nothing.

D. Neither changes; calibration only relabels the axis of the reliability diagram.

5 Why does the course prefer WAPE over MAPE for scoring a demand forecast?

A. It squares the errors, so a single large miss dominates the score.

B. It divides total absolute error by total actual volume, so small actuals cannot dominate.

C. It compares the model's error against the naive last-value forecast.

D. WAPE is expressed without units, whereas MAPE is expressed in the units of the quantity being forecast.

6 A retrieval-augmented assistant answers 62% of questions correctly. On the same questions, recall@5 is 0.55. Where does the course say to look first?

A. The prompt, since the model is evidently failing to use the context it is given.

B. The judge, since a 62% figure may be a scoring artefact rather than a real rate.

C. Temperature, since lowering it raises groundedness and therefore accuracy.

D. Retrieval, since the generator cannot recover an answer it was never handed.

MODULE 3

Deciding whether a difference is real

Two systems, two numbers, and the question that decides the week: is the gap real, or is it the sample? This block does the arithmetic — a paired test you can compute on paper, a bootstrap in ten lines, the number of items a comparison actually needs — and the four ways a comparison lies to you before you have looked at the model at all.

> BY THE END OF THIS MODULE YOU CAN
>
> Decide whether an observed difference between two systems is larger than the noise in your set, compute an interval for any metric you can calculate using the bootstrap, and state in advance how many items the comparison needs to detect the effect you would act on

21 · 9 min

Asking what pure chance would have produced

THE ONE THING TO KEEP

Shuffle which system produced which result ten thousand times and see how often chance beats your observed gap — one loop replaces a shelf of named tests and encodes your design honestly.

The question under every comparison

Prompt B scores 5 points above prompt A on the same 200 items. Before anything else, one question has to be answered: if the two prompts were identical in quality, how often would a gap that big appear anyway?

There are two ways to answer it. You can look up the right named test, check its assumptions, and hope you chose correctly. Or you can build the world where the null is true and count. The second is a permutation test, it takes ten lines, and it is very hard to get wrong in a way you would not notice.

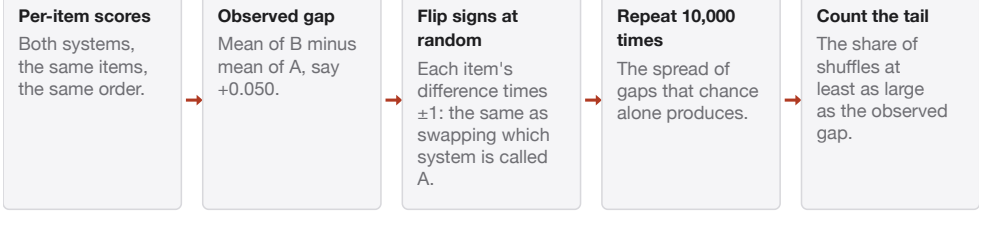
The engine

You have per-item results for both systems. Under the null hypothesis, the label saying which result came from A and which from B carries no information — so swapping them within an item should change nothing systematic. Do exactly that, many times.

```
import numpy as np
rng = np.random.default_rng(0)

a = np.array(scores_a)          # one score per item, same or-
der
b = np.array(scores_b)
observed = b.mean() - a.mean()  # e.g. +0.050

diffs = b - a
null = []
for _ in range(10_000):
    signs = rng.choice([-1, 1], size=len(diffs))
    null.append((diffs * signs).mean())

null = np.array(null)
p = (np.abs(null) >= abs(observed)).mean()
print(f"gap {observed:+.3f}, p = {p:.3f}")
```

Flipping the sign of each item's difference is the same as swapping which system is called A. Run it and you get a distribution of gaps that chance alone produces — for 200 items with this much spread, typically a standard deviation around 0.021. Against that, an observed +0.050 sits well out in the tail, and roughly 1.5% of shuffles exceed it. That is your p-value, computed from your own data, with no assumption of normality and no table.

A permutation test in five steps



For 200 items the null gaps spread about ±0.021 either way, so an observed +0.050 is beaten by roughly 1.5% of shuffles. That share is the p-value, with no table and no normality assumption.

Why this beats remembering tests

The permutation approach works for any statistic you can compute. Median latency. Recall at 5. A weighted composite of three rubric checks. Cost per successful task. There is no closed-form test for most of these, and you do not

need one — you only need to be able to compute the number and to know what the null would scramble.

It also forces you to state your design, which is the part that most often goes wrong. **What you shuffle is your claim about what is exchangeable.**

- Paired evaluation, same items through both systems: flip signs within each item, as above.
- Two separate groups of users: shuffle the group labels across users.
- Items grouped by customer: shuffle at the customer level, keeping each customer's rows together.

Shuffling rows when the rows are grouped gives a confidently wrong answer, and the next lessons cover why. The virtue of writing the shuffle by hand is that the mistake becomes visible in code rather than hiding in a function's assumptions.

The A/A test, which you should run first

Before trusting any comparison machinery, point it at two copies of the same system. Run configuration A twice, feed both result sets into the test, and see what it says.

The result should be an unremarkable gap and a p-value scattered anywhere between 0 and 1. If A against A produces a "significant" difference, you have found a bug, and it is nearly always one of five:

- The two runs used different subsets, because an item errored out in one of them and was silently dropped.
- Results were compared in the wrong order, so item 41's A score sits against item 42's B score.
- A cache served the first run and not the second.
- The set is ordered, and a truncated run covers a different mix.
- Non-determinism is larger than you assumed, which is not a bug but is a finding.

An A/A test costs one extra run and it has caught bad harnesses at essentially every team that has bothered.

What it cannot do

A simulation answers exactly the question you encoded, and nothing more.

- It cannot rescue a biased sample. If your 200 items over-represent one intent, permutation gives you a precise p-value about a set that does not resemble your traffic.
- It cannot tell you the effect is important. A gap can be reliably real and commercially worthless; report the size and its interval next to the p-value.
- It assumes the outcomes you fed it are comparable. Two runs at temperature 0.7, compared without accounting for run-to-run variance, will produce a small p for a difference that vanishes tomorrow.

Ten lines, `numpy`, and a laptop. For small sets you can do the same thing in a spreadsheet with a column of random signs, and it is worth doing once by hand to see the null distribution appear.

BEFORE YOU MOVE ON

A team runs a permutation test by shuffling all 600 result rows freely between the two systems, but the 600 rows come from 120 customers with 5 rows each. What does that shuffle get wrong?

A. Nothing, provided the number of rows assigned to each system stays equal after each shuffle.

B. It breaks the pairing, so the test loses power and will fail to detect a real difference.

C. It treats correlated rows from one customer as independent, so the null distribution is narrower than reality and the p-value comes out too small.

D. It requires the two systems to have been run on identical inputs, which grouped data cannot guarantee.

22 · 9 min

What a p-value is, and the four things it is not

THE ONE THING TO KEEP

A p-value is the chance of seeing a gap this large if there were no effect — it is not the probability you are right, and when few of your ideas work most of your significant results are still wrong.

The definition, stated carefully

A p-value is the probability of observing a result at least as extreme as the one you got, **assuming the null hypothesis is true**. That is the entire content. Everything difficult about p-values comes from that conditional clause, which people drop within seconds of reading it.

Four things it is not:

- **Not the probability that the null is true.** A p of 0.03 does not mean a 3% chance the systems are equal, nor a 97% chance B is better. Getting from one to the other needs a prior, which the p-value does not contain.
- **Not a measure of size.** A tiny p can accompany a difference of 0.2 points.
- **Not a measure of importance.** Those are different words for a reason.
- **Not repeatable.** A p-value is itself a random quantity. Rerun a genuine experiment with 50% power and the p bounces between 0.001 and 0.4 across replications. Treating 0.049 and 0.051 as different states of the world is a category error.

The arithmetic that changes how you ship

Here is the calculation that matters most for a team pushing prompt changes, and it is rarely done.

Suppose you try 100 prompt variations over a quarter. Realistically, perhaps 10 of them genuinely help. Your tests use the usual 5% threshold, and with your set size you have about 50% power to detect the effects you care about.

- Of the 10 true improvements, you detect **5**.
- Of the 90 changes that do nothing, 5% come out significant anyway: **4.5** false alarms.

So you finish the quarter with about 9.5 "wins", of which nearly half are noise, and you shipped all of them. Nothing was done incorrectly; each individual test behaved exactly as advertised. The false-discovery rate depends on how often your ideas are right, and nobody's ideas are right most of the time.

Three consequences follow:

1. **Raise power before lowering alpha.** Going from 50% to 80% power turns 5 true detections into 8 and leaves the 4.5 false alarms unchanged.
2. **Replicate anything surprising.** A second, independent run on fresh items costs a day and removes most of the false alarms, because a fluke rarely repeats.
3. **Prefer changes with a mechanism.** A change you can explain is drawn from a population with a much higher true-positive rate than a change found by trying things.

Significant and worthless

Online, the opposite failure dominates. With 200,000 sessions, a difference of 0.15 percentage points in click-through will be significant at any threshold you like. The correct response is not to celebrate; it is to ask whether 0.15 points is worth the maintenance cost of the change, the extra latency, and the new failure mode.

This is why the pre-registered decision threshold from earlier in the course exists. Write down before the experiment: "we will ship if the improvement is at least 2 points, and not otherwise". Then significance becomes a check on whether you can see 2 points, rather than the decision itself.

Non-significant does not mean no difference

The most consequential misreading in practice. "p = 0.31, so the models are the same" is wrong; the correct statement is that the experiment could not distinguish them. Report the interval instead:

> *B is 2 points better (95% CI −3 to +7). We can rule out a regression larger than 3 points and an improvement larger than 7.*

That sentence supports a decision. "Not significant" does not. If the interval is so wide that both a serious regression and a large gain are inside it, the honest conclusion is that you ran the wrong-sized experiment, and the next lesson is about avoiding that.

Where 0.05 came from

Fisher suggested it in the 1920s as a convenient convention and said explicitly that no fixed level should be used for all purposes. It has no mathematical standing. Choose from consequences instead:

- Changing a prompt on a low-risk internal tool: 0.10 is fine; the cost of a wrong call is a revert.
- Changing a safety threshold: use a much stricter level, several independent runs, and human review regardless of the number.

State the level you chose, before you look, and do not change it afterwards. Moving the threshold after seeing the result is how a decision procedure quietly becomes a story.

BEFORE YOU MOVE ON

A team runs 100 prompt experiments in a quarter at the 5% level with roughly 50% power, and about one in ten of their ideas genuinely helps. Roughly what share of their significant results are false alarms?

A. About 5%, since that is the significance level they chose.

B. About 50%, because half the true effects are missed and the missed half is replaced by false positives.

C. Roughly 4.5 false alarms against 5 true detections, so nearly half.

D. None, provided each experiment was correctly designed and pre-registered.

23 · 9 min

Sizing the set before you run it

THE ONE THING TO KEEP

Start from the smallest difference worth acting on, not the one you hope for, and compute the item count it needs — for a five-point gap around 80% that is roughly a thousand items per side unless the comparison is paired.

Ask it in the other direction

The usual question is "we have 200 items, what can we conclude". The useful question is asked before the run: "what is the smallest difference I would act on, and how many items does seeing it require".

That first number is the minimum detectable effect, and it is a product decision, not a statistical one. Would you ship for 1 point? For 3? If a 2-point improvement would not change what you do, do not design an experiment to find one.

The arithmetic for two independent groups

For comparing two rates around `p`, with the conventional 80% power and 5% significance, the sample size per group is close to:

```
n ≈ 16 * p * (1 − p) / d²
```

where `d` is the difference you want to detect. Put in real numbers for a system passing about 80% of the time, hoping to detect 5 points:

```
n ≈ 16 * 0.8 * 0.2 / 0.05²  =  16 * 0.16 / 0.0025  =  1,024 per group
```

Two thousand labelled items to resolve five points. That number surprises people and it is the honest arithmetic. Halve the effect you want to detect and it quadruples, because `d` is squared: detecting 2.5 points needs about 4,100 per group.

This is why most published comparisons on a hundred hand-labelled items cannot support the claims made from them, and why the next section matters so much.

Pairing changes everything

Running both systems on the *same* items removes the item-difficulty variation that dominates the noise above. What is left is the flips: items where the two systems disagree.

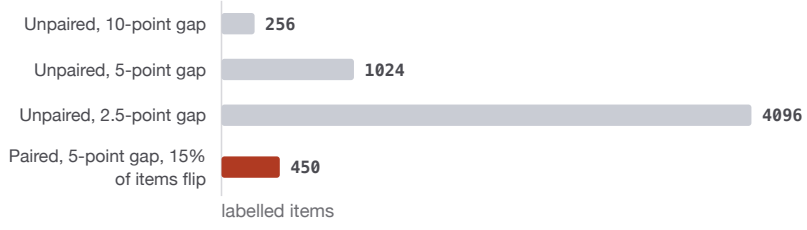
The count that matters is the number of discordant pairs, and it is much smaller than the total.

- To detect that flips run 2 to 1 in your favour rather than evenly, you need roughly **68 discordant pairs**.
- For a starker 3 to 1 split, about **29**.

If a change flips 15% of items, 68 discordant pairs means about 450 items. If it flips 30%, about 230. So a paired design turns a 2,000-item requirement into

a few hundred — the single largest efficiency available in evaluation, and the reason the harness must record per-item results rather than totals.

Items needed to detect a gap around an 80% pass rate



n ≈ 16 p(1 − p) / d² at 80% power and 5% significance, per group. Halving the gap quadruples the count. Pairing counts only the discordant items, so about 68 flips settle the same five-point question.

When you cannot remember any formula

Simulate. This generalises to any metric and any design, and you can check it against your own data.

```
import numpy as np
rng = np.random.default_rng(0)

def power(n, p_a=0.80, effect=0.05, trials=2000):
    wins = 0
    for _ in range(trials):
        a = rng.random(n) < p_a
        b = rng.random(n) < p_a + effect
        se = np.sqrt(p_a * (1 - p_a) * 2 / n)
        wins += abs(b.mean() - a.mean()) > 1.96 * se
    return wins / trials

for n in (200, 500, 1000, 2000):
    print(n, round(power(n), 2))     # 0.22  0.47  0.77  0.97
```

Two hundred items gives you a 22% chance of detecting a real five-point gain. Four times in five you would conclude nothing and move on, having thrown away a genuine improvement.

The simulation is also where you discover the second cost, which no formula shows: labelling. Two thousand items at two minutes each is over sixty hours

of somebody's attention, and that is the real reason evaluation sets are small. Knowing the arithmetic lets you spend that budget where it buys the most — usually on the one comparison that decides an architecture, rather than spread thinly across every prompt tweak.

The effect you plug in is not the effect you hope for

Power calculations are routinely run with the improvement somebody wants rather than the one that would change a decision, and the difference is not academic. Put in 10 points because that is the target, and the arithmetic returns a comfortable 260 items per group; the experiment then has almost no chance of detecting the 4-point gain that actually arrives, and reports nothing. Put in the smallest difference worth acting on, get a larger and more honest number, and decide from there whether to fund it.

What to do when you cannot afford it

You will often be unable to label 2,000 items. Four honest responses, in order of preference:

1. **Pair the comparison** and count flips, as above.
2. **Use a continuous per-item score** instead of pass or fail. A binary throws away magnitude; a 0-to-4 rubric total carries more information per item and needs fewer of them.
3. **Reduce variance** by stratifying and by fixing seeds — the subject of a later lesson in this block.
4. **Accept the limit and write it down.** "This set can detect a 12-point difference. It cannot resolve the 4-point gap between these two prompts." That sentence is a result. It prevents a false claim, and it makes the case for the budget to label more.

The one thing not to do is run the small set, get a non-significant answer, and keep adding items until it turns significant. That converts a controlled error rate into an uncontrolled one, and it is the subject of the peeking lesson later in this block.

BEFORE YOU MOVE ON

A team wants to detect a 2-point improvement on a system that currently passes 80% of items, using two independent sets. They have budget to label 600 items in total. What is the defensible plan?

A. Label the 600 items, run the comparison, and report whatever comes out with its confidence interval.
B. Run both systems on the same 600 items and analyse the flips, accepting that only a difference producing a clear imbalance in discordant pairs will be detectable.
C. Split into three sets of 200 and take the majority verdict across the three comparisons.
D. Use a 10% significance level instead of 5%, which roughly doubles the power at this sample size.

24 · 9 min

The bootstrap, and the three places it lies

THE ONE THING TO KEEP

Resampling can only ever redraw values you already observed, so it gives honest intervals for means and rates and badly overconfident ones for maxima, extreme percentiles and clustered data.

What resampling actually assumes

The bootstrap treats your sample as a miniature of the population: draw from it with replacement, recompute the statistic, and the spread of those recomputations approximates the spread you would see across genuinely new samples.

That assumption is reasonable for a mean or a rate on a decent-sized independent sample, which is why the five-line bootstrap covers most of the intervals

you will ever need. It fails in three specific ways, each with a mechanism you can state.

One: it cannot see past the largest value you have

Resampling draws from the values you already hold. The largest possible bootstrap maximum is your observed maximum, so an interval for a maximum is bounded by data rather than by the world.

The practical version of this is tail latency. Bootstrap p99 from 200 requests and the estimate is computed from the two slowest requests you happen to have. The interval will look tight and it is meaningless: the next 200 requests will contain a 9-second outlier your sample never saw.

Rules that follow:

- Do not bootstrap a maximum, a minimum, or a percentile beyond roughly `1 − 5/n`. With 200 items, p95 is already shaky and p99 is not measurable.
- For tails, get more data or model the tail's shape explicitly. Ten thousand requests give a p99 computed from a hundred observations, which is a real estimate.

Two: it inherits any dependence in your rows

Bootstrap rows drawn independently from a set whose rows are not independent produces intervals that are too narrow — sometimes by a factor of two. Five responses per user, or five runs of the same item, are not five independent observations. The next lesson deals with this properly; the rule here is that whatever the unit of independence is, that is the unit you resample.

```
# resample customers, take all of each customer's rows
groups = df.groupby("customer_id").indices
keys = list(groups)
draw = rng.choice(keys, size=len(keys), replace=True)
sample = df.iloc[np.concatenate([groups[k] for k in draw])]
```

Three: small samples and lumpy statistics

Below roughly 30 items the bootstrap's coverage degrades — the 95% interval contains the truth noticeably less than 95% of the time, because a sample that small is a poor miniature of anything. And for statistics that move in jumps, such as a median over integer rubric scores, the bootstrap distribution becomes a few spikes rather than a smooth spread, so the percentiles land on arbitrary points.

For those cases prefer the mean of a per-item score over the median, or use an exact method, or state a range rather than an interval.

Percentile intervals and the better default

The simple approach — take the 2.5th and 97.5th percentiles of the bootstrap distribution — is fine when the statistic's distribution is roughly symmetric. When it is skewed, which is normal for ratios and for metrics near 0 or 1, the interval is shifted. The bias-corrected and accelerated (BCa) interval corrects for that, and `scipy` does it for you:

```
from scipy.stats import bootstrap
res = bootstrap((scores,), np.mean, n_resamples=10_000,
                confidence_level=0.95, method="BCa", random_s-
tate=0)
print(res.confidence_interval)
```

Use 1,000 resamples while exploring and 10,000 for a number you will publish; the Monte Carlo noise in the interval's endpoints shrinks with the square root of that count, and 1,000 leaves visible jitter in the third digit.

For comparisons, resample once

The most common mistake in a paired comparison is bootstrapping each system separately and comparing the two intervals. Overlapping intervals do not mean no difference, because the two systems' errors are correlated — they were measured on the same items.

Resample the item indices once, score both systems on that resample, and record the difference. The interval you want is the interval of the difference. It is routinely half the width of what the two separate intervals suggest, and it is the correct one.

The habit worth keeping

Store per-item results, always. Every technique in this module — permutation, bootstrap, paired analysis, slicing — needs the vector of per-item outcomes and can do nothing with a summary. A harness that logs "84%" and discards the rows has thrown away every question you will want to ask next week.

BEFORE YOU MOVE ON

A team bootstraps the 99th percentile of response latency from 300 logged requests and reports a 95% interval of 4.1 to 4.4 seconds. Why is that interval untrustworthy?

A. Latency is skewed, so a percentile interval is invalid and BCa should have been used instead.

B. p99 from 300 requests is computed from about three observations, and resampling can only redraw values already present, so the interval reflects those three points rather than the tail.

C. Three hundred requests is below the threshold where the bootstrap has adequate coverage.

D. The bootstrap assumes the metric is normally distributed and latency is not.

25 · 9 min

Five hundred rows, forty users

THE ONE THING TO KEEP

When rows share a source — the same user, the same document, repeated runs of the same item — your effective sample size is closer to the number of sources than the number of rows, so aggregate to the source before you compute anything.

The interval that keeps being wrong

A team measures a support assistant on 500 logged conversations. The pass rate is 84%, the reported interval is plus or minus 3 points, and yet each fresh sample lands 6 or 7 points away. Nothing is broken in the arithmetic. The 500 conversations came from 40 customers, and rows from one customer resemble each other.

Every standard error formula assumes independent observations. Correlated rows carry less information than independent ones, and the amount less has a name.

The design effect, in numbers

With `m` rows per cluster and an intra-cluster correlation `ICC`, the variance is inflated by:

```
design effect = 1 + (m − 1) ∗ ICC
```

For 500 rows from 40 customers, `m` is 12.5. Suppose the ICC is 0.3, which is unremarkable for text from the same organisation:

```
design effect = 1 + 11.5 ∗ 0.3 = 4.45
effective n   = 500 / 4.45     ≈ 112
```

You have 500 rows and about 112 rows' worth of information. The honest interval is `sqrt(4.45)` — a bit over twice — wider than the one printed, which is precisely the discrepancy the team kept seeing.

The same arithmetic covers the other common case: 100 items run 5 times each at temperature 0.7. Those 500 rows are not 500 independent observations either, and the clusters are the items.

Three fixes, in order of preference

Aggregate to the cluster. Compute the rate per customer, then average across customers, and do all the statistics on those 40 numbers. It is simple, it is safe, and every formula becomes valid again because the 40 values are independent. The cost is a wider, correct interval.

```
per_customer = df.groupby("customer_id")["passed"].mean()
n = len(per_customer)                                  # 40, not
500
se = per_customer.std(ddof=1) / np.sqrt(n)
```

Cluster bootstrap. Resample customers with replacement, carrying all of each customer's rows, and recompute. Use this when the statistic is not a simple mean — recall at 5, a weighted composite, a per-item pass rate you want to weight by volume.

Cluster-robust standard errors or a mixed model. `statsmodels` will fit both. Worth it when you need to adjust for covariates at the same time; unnecessary otherwise.

Finding the clusters, which are rarely labelled

The customer column is the easy case. The clusters that cause trouble are the ones nobody wrote down:

- **One document, many questions.** Forty questions generated from the same PDF share its formatting, its vocabulary and its ambiguities. The unit is the document.

- **One template, many variants.** Everything the previous module said about synthetic items applies here: forty rephrasings of one request are one observation wearing forty coats.
- **One session, many turns.** A conversation that goes wrong at turn two produces five failed turns, which is one failure.
- **One time window.** Everything logged during an outage shares the outage.

Before computing an interval, ask what the rows have in common that the population would not. If the answer is "quite a lot", find the grouping key and aggregate to it.

Repeated runs deserve their own decomposition

When each item is run several times, two different variances are in play and they suggest different actions:

- **Item-to-item variance** — some items are simply harder. More runs will not reduce it; more *items* will.
- **Run-to-run variance** — the same item passing on Tuesday and failing on Wednesday. More items will not reduce it; more runs, a lower temperature, or a fixed seed will.

Compute both. Take the per-item mean across runs, and look at the spread of those means (item variance) alongside the average spread within an item (run variance). If run-to-run spread is 5 points, then no amount of extra data makes a 3-point improvement detectable, and the correct next step is to make the system more deterministic rather than to label more.

Say both numbers out loud

The habit that prevents all of this is one line in every report: state the number of rows and the number of independent units.

> *Pass rate 84% across 500 conversations from 40 customers (95% CI 76–92, computed at the customer level).*

A reader who sees "40 customers" immediately knows the resolution available. A reader who sees only "500 conversations" will believe the study is four times more precise than it is.

The limit you cannot compute your way out of

If the set contains 8 customers, no statistical technique makes it behave like 8,000. Clustering is a data-collection problem with a statistical symptom. When the effective sample is small, the truthful report says so and recommends widening collection — more customers, more documents, more languages — rather than quoting a narrow interval that the next sample will contradict.

BEFORE YOU MOVE ON

An evaluation shows 84% on 500 conversations drawn from 40 customers. Which reported figure is most defensible?

A. 84% with a 3-point interval, since 500 items is a large sample by evaluation standards.

B. 84% with no interval, since clustered data cannot support a confidence interval.

C. 84% averaged across customers, with the interval computed from the spread of the 40 customer-level rates.

D. 84% with the interval computed after removing the customers contributing the most conversations, so no single customer dominates.

26 · 8 min

Twenty slices and one guaranteed surprise

THE ONE THING TO KEEP

Testing twenty slices at the five per cent level makes a false alarm more likely than not, so name one primary metric in advance and treat every slice as a lead to replicate rather than a result to ship.

The arithmetic of looking everywhere

A change ships. You slice the result by language, device, tenure, plan, region, channel, input length, and intent — twenty slices. One of them shows a significant regression. The team spends two days on it.

Compute the chance of that happening when the change did nothing at all:

```
P(at least one false alarm) = 1 - 0.95 ** 20 = 0.64
```

Nearly two times in three you will find a "significant" slice in a change with no effect whatsoever. Not because anybody cheated — because twenty independent chances at a one-in-twenty event is what that expression says.

The same arithmetic applies to comparing eight prompt variants against a control, to tracking fifteen metrics on one experiment, and to running the same test weekly for a quarter.

Three corrections, and when each fits

Bonferroni. Divide the threshold by the number of tests: with 20 slices, require p below 0.0025. It is correct, it is one line, and it is severe — power collapses, and you will miss real regressions in small slices. Use it when a false alarm is expensive and the tests are few.

Holm. Sort the p-values, compare the smallest against `alpha/m`, the next against `alpha/(m−1)`, and stop at the first failure. It controls the same error rate as Bonferroni and is uniformly more powerful, so there is no reason to prefer plain Bonferroni once you know it exists.

Benjamini–Hochberg. Controls the false discovery rate — the expected share of your claims that are wrong — rather than the chance of any error at all. Sort the p-values ascending and find the largest `i` with:

```
p(i) <= (i / m) * q          # q = 0.10 is a common choice
```

Reject everything up to that rank. This is the right tool for exploratory slicing, where you want a list of leads with a known contamination rate rather than a guarantee of purity.

```
from statsmodels.stats.multitest import multipletests
reject, p_adj, _, _ = multipletests(pvals, alpha=0.10,
method="fdr_bh")
```

Worked through: with 20 slices and q = 0.10, a p-value of 0.004 sitting at rank 1 passes because `1/20 * 0.10 = 0.005`, while a p-value of 0.03 at rank 5 fails because `5/20 * 0.10 = 0.025`. Bonferroni would have rejected both. The gain is real and it is bought by accepting that roughly one in ten of your accepted findings is noise, which is fine for a list of leads and not fine for a shipping decision.

The correction is not the real fix

Corrections manage the arithmetic; they do not manage the behaviour that creates it. Two habits matter more.

Name the primary metric before the run. One metric, one slice definition, one threshold, written down. Everything else in the report is labelled exploratory. This single convention removes most multiplicity problems, because the decision now rests on a test that was specified before any data existed.

Replicate anything surprising on fresh data. A real regression in Hindi shows up again next week. A fluke does not. One replication is worth more than any correction, and it costs a rerun.

You do not need twenty tests to have the problem

The subtler version has no visible count. You look at the results, notice that long inputs seem worse, and test that. You ran one test — but you chose it after seeing the data, from among the dozens of slices you might have noticed. The error rate belongs to the set of things you *could* have chosen, not the one you did. This is why exploratory analysis is a source of hypotheses, and hypotheses need their own data.

The practical version: when a slice finding is found by looking, collect fresh items for that slice and re-measure. Frequently the effect halves, which is itself informative.

Guardrails go the other way

There is one deliberate exception. For metrics protecting against harm — safety violations, refusals, severe latency, error rates — you *want* a sensitive tripwire, and you accept more false alarms to get it. Do not apply a multiplicity correction to a guardrail. Set it loose, investigate every alert, and accept that most investigations find nothing. The asymmetry is intentional: an unnecessary investigation costs an afternoon, and a missed safety regression costs something you cannot buy back.

Write both policies into the report template, so nobody has to argue about it while looking at a fresh number: primary metric with its pre-registered threshold, exploratory slices with FDR control, guardrails uncorrected and always investigated.

BEFORE YOU MOVE ON

A team reports that a new prompt significantly improved one of eighteen slices they examined at the 5% level, and proposes shipping on that basis. What is the strongest objection?

A. The slice was probably too small to support a significant result, so the finding is driven by a handful of items.

B. Eighteen tests at 5% give roughly a 60% chance of at least one false positive, so the finding is a lead that needs replication on fresh data rather than a result.

C. Slice-level results should never be used for decisions, only whole-set results.

D. They should have applied a Bonferroni correction, which would have made the result significant at the corrected level.

27 · 9 min

Watching the dashboard until it says yes

THE ONE THING TO KEEP

Checking an experiment repeatedly and stopping when it crosses the line turns a five per cent error rate into something far larger, and the effect you stop on is overestimated because you stopped at a high point.

What repeated looking does

An online experiment runs for a fortnight. The team checks the dashboard every morning and will stop when the p-value drops below 0.05. This feels efficient and it silently destroys the error rate.

The mechanism is simple. A p-value computed on accumulating data wanders. Even when there is no effect at all, it drifts up and down, and over enough looks it will dip below 0.05 at some point. Checking five times pushes the false-positive rate from 5% to roughly 14%. Checking continuously, with

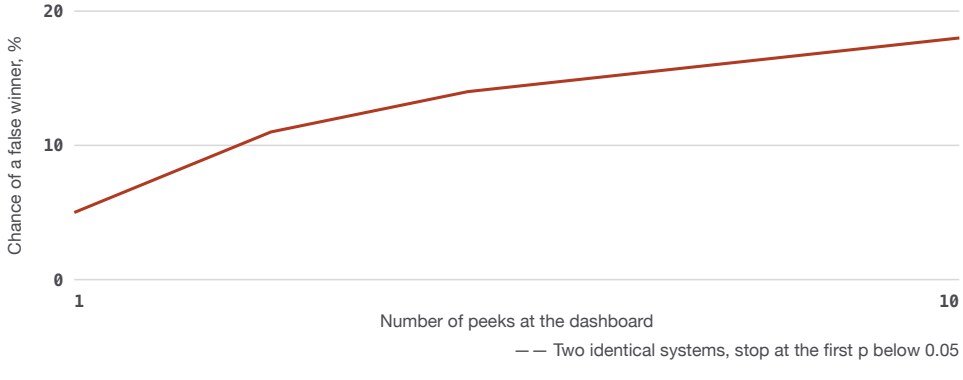
no fixed horizon, means the probability of eventually crossing approaches certainty.

You can watch it happen in a few lines:

```
import numpy as np
rng = np.random.default_rng(0)
hits = 0
for _ in range(2000):
    a = rng.random(4000) < 0.20        # identical systems
    b = rng.random(4000) < 0.20
    for n in range(400, 4001, 400):    # ten peeks
        pa, pb = a[:n].mean(), b[:n].mean()
        se = np.sqrt(2 * 0.2 * 0.8 / n)
        if abs(pb - pa) > 1.96 * se:
            hits += 1
            break
print(hits / 2000)                      # ≈ 0.18
```

Two identical systems, and eighteen per cent of experiments declare a winner.

The false-positive rate rises with every look



The nominal 5% holds for exactly one look. Ten looks at an experiment with no effect at all declare a winner about one time in five, and the effect reported at the crossing is biased upwards.

The second, quieter damage

Stopping at the moment of crossing does not only produce false positives. It biases the *size* of every effect you report, including the real ones.

You stop when the estimate is unusually high, because that is what crossing the threshold means. The estimate at that moment is therefore drawn from the upper part of its own distribution. This is the winner's curse, and it is why so many shipped improvements shrink when re-measured: a 4-point gain declared on day three becomes 1.5 points when the experiment is allowed to run out.

If you have stopped early, say so and treat the effect size as an upper bound until it has been re-measured on new data.

What to do instead

Fix the horizon. Decide the sample size in advance using the power arithmetic from earlier in this block, decide the end date, and look once when it arrives. This is the simplest correct design and it costs nothing but patience.

If you must look, budget for it. Group sequential methods spend the error rate across a planned number of looks. O'Brien–Fleming boundaries are strict early and relax towards the end, so an early stop requires overwhelming evidence; Pocock boundaries are constant and stop earlier at the cost of a stricter final threshold. Both require the number and timing of looks to be planned in advance.

Or use a method built for continuous monitoring. Sequential tests based on confidence sequences — sometimes marketed as "always-valid" p-values — give an interval that is correct at every moment you choose to look. The price is width: the interval is meaningfully wider than a fixed-horizon interval on the same data, which is exactly the cost of the freedom to peek. Several open-source implementations exist, and most mature experimentation platforms now ship one.

There is a third option nobody names, and it is often the right one: **stop looking**. Most experiments run for a fortnight because a week of traffic is needed, not because anybody will act mid-flight. Hiding the dashboard until the end date removes the temptation entirely, costs nothing, and is easier to enforce than a boundary nobody remembers the rationale for. Where a team insists on visibility, show the sample size accumulating and the guardrails, and withhold the primary metric until the horizon.

The offline version of the same sin

This module's arithmetic applies to eval sets too, in a form that looks innocent: run the set, get a non-significant result, add fifty more items, check again, repeat until it turns significant. That is optional stopping with extra steps.

Run the whole set. If it is inconclusive, decide the new size in advance, collect it, and rerun the whole comparison on the enlarged set — reporting that you did so.

One exception, and it is deliberate

Stopping early for harm is not only permitted, it should be planned. Write the guardrail conditions before launch — a safety-violation rate above some level, an error rate above another, a latency regression beyond a stated bound — and stop the moment they trigger, without a significance test.

The asymmetry is on purpose. Stopping early for a good result buys you a few days and costs you a reliable number. Stopping early for a bad result costs you a few days and protects users. Those are not the same trade, and the stopping rule should not treat them as though they were.

BEFORE YOU MOVE ON

An experiment is checked daily and stopped on day four when the improvement first reaches significance at 6.2 points. What is the most likely relationship between 6.2 points and the true effect?

A. It is an unbiased estimate, since the significance test already accounts for how much data was collected.

B. It understates the true effect, because early data has more noise and noise pulls estimates towards zero.

C. It overstates the true effect, because stopping was triggered by the estimate being unusually high at that moment.

D. It is unrelated to the true effect, since the daily peeking has invalidated the measurement entirely.

28 · 9 min

Getting more signal from the items you already have

THE ONE THING TO KEEP

Fix every source of noise the change did not cause — same items, same seeds, same retrieval snapshot, plus a covariate adjustment — and a set you cannot afford to enlarge starts detecting differences half the size.

The other lever

Sample size is one way to see smaller differences. Reducing the noise in each observation is the other, and it is usually cheaper. Every technique here works by removing variation the change did not cause, so that what remains is the change.

Hold everything else still

The largest single gain is running both systems on the same items, which the course has already established. The rest of the principle is the same idea applied to everything else that wobbles:

- **Fix seeds and set temperature to 0** for the comparison run, even if production runs hotter. You are measuring a difference between configurations, not simulating production.
- **Freeze the retrieval corpus.** A vector index that ingested 2,000 new documents overnight makes yesterday's numbers incomparable, and this is the most common cause of a mysterious drift in a RAG evaluation.
- **Pin the model version and the tokeniser**, and record both.
- **Use the same ordering and the same batching**, since provider-side batching can change outputs subtly.

This is the common-random-numbers idea from simulation: when two systems share their random draws, the shared part cancels in the difference and the comparison gets sharper. It costs nothing but discipline.

Stratify

If a known factor drives most of the variance — language, input length, intent — allocate the set proportionally across it and compute a stratified estimate rather than a single average. The between-stratum variance disappears from the estimate's error, leaving only the within-stratum part. When language accounts for, say, 40% of the total variance, stratifying removes that 40% from the noise for free.

Adjust with a covariate

This is the technique most teams have never tried offline, and it is the strongest one available.

If you have any per-item number that correlates with the outcome but is unaffected by your change, you can subtract its influence. Online, that number is usually the user's behaviour in a pre-experiment period — the method known as CUPED. Offline, it can be the score of a cheap baseline on the same item, an item difficulty rating, or the previous release's score.

```
Y_adjusted = Y − theta ∗ (X − mean(X))       theta = cov(Y, X)
/ var(X)
```

The adjusted metric has the same expectation and a variance reduced by a factor of `1 − rho²`, where `rho` is the correlation between `Y` and the covariate. At `rho = 0.7` that is a 51% reduction — the same precision as doubling the number of items, at the cost of eight lines of code.

Use a score with more information in it

A pass or fail throws away magnitude. An answer that misses one required field and one that is entirely wrong both score 0, and the difference between them is invisible to the metric. A graded rubric — four binary checks summed

to a 0-to-4 score — carries more information per item and needs fewer items for the same resolution. This is one reason the rubric lesson later in the course insists on several small checks rather than one holistic rating.

Repeats, and a result that surprises people

Suppose the model is non-deterministic, so each item has run-to-run noise on top of item-to-item noise. With `n` items and `k` runs each, the variance of the overall mean is:

```
(item_variance + run_variance / k) / n
```

Fix a budget of `n * k` calls. Substituting shows the variance is minimised at `k = 1`: for estimating an average, one run on many items beats several runs on few. Repeats buy precision only on the individual item, not on the mean.

So use one run per item when the question is "which system is better overall", and use repeats when the question is "is this specific item reliably passing" — a regression suite, a safety case, a customer's reported bug. Two different questions, two different designs, and teams routinely spend their budget on the wrong one.

What none of this fixes

Variance reduction makes an estimate more precise. It does nothing about bias. A set drawn from the wrong traffic, labelled against a confused guide, or contaminated by training data will give you a tighter interval around the wrong number — which is worse than a loose interval around it, because it is more convincing. Fix the sampling and the labels first, then sharpen.

BEFORE YOU MOVE ON

A team has a fixed budget of 1,200 model calls to compare two prompts, and the model is non-deterministic. Which allocation gives the most precise estimate of the difference in mean quality?

A. 1,200 items run once each, since for estimating a mean the run-to-run noise is divided by the item count anyway.
B. 400 items run three times each, because averaging three runs per item removes the non-determinism before the comparison.
C. 240 items run five times each, so every item's score is stable enough to trust individually.
D. 600 items run twice each, as a compromise between item coverage and run stability.

29 · 8 min

Better in every group and worse overall

THE ONE THING TO KEEP

An overall score is a weighted average, so a change in the mix of hard and easy items can reverse it while every group improves — fix the weights before comparing two numbers.

The reversal, in numbers

Two models, measured on Hindi and English requests.

The old model was measured on 500 Hindi items and 500 English items: 68% on Hindi, 89% on English. Overall, 78.5%.

The new model was measured on 800 Hindi and 200 English: 72% on Hindi, 91% on English. Overall:

```
0.8 * 72 + 0.2 * 91 = 57.6 + 18.2 = 75.8%
```

The new model is better in Hindi, better in English, and 2.7 points worse overall. Nobody made an arithmetic error. The two overall numbers are weighted averages with different weights, and Hindi is the harder half.

Where the different weights come from

This is not a curiosity. Every one of these produces it:

- **The set was refreshed between measurements.** Traffic grew in one market, so the new sample contains more of it.
- **Assignment correlates with a segment.** The new experience shipped to the app first, and app users differ from web users.
- **A router sends hard cases to the new system.** Any escalation rule guarantees the new system is measured on a harder mix.
- **A canary is regional.** The 5% rollout went to one data centre, which serves one part of the world.
- **Volume itself moved during the experiment.** A campaign, an outage, a public holiday.

The last one is the reason a production metric can move several points with no change to the model at all. That is a mix shift, and the first question when a dashboard drops should be "did the mix move" rather than "what broke".

The fix is to fix the weights

Report per slice, always. Then, if a single number is required, compute it against a fixed reference population — direct standardisation:

```
ref = {"hi": 0.55, "en": 0.45}                          # frozen
traffic mix
overall = sum(ref[s] * rate[s] for s in ref)
```

Both systems are now scored on the same imaginary population, so the comparison is valid. Write the reference weights into the report and version them; when the real mix moves far enough that the reference is fiction, update it deliberately and re-baseline everything, rather than letting it drift silently.

The same trick answers the monitoring question. Track the raw metric and the mix-standardised metric side by side. When the raw one moves and the standardised one does not, your users changed rather than your system.

There is one more guard worth building into the harness: refuse to print a single overall number when the slice proportions of the two runs differ by more than a few points. A comparison whose weights moved is not a comparison, and a harness that says so is more useful than one that quietly averages. Ten lines of code, and it catches the case where somebody re-sampled the eval set between two runs and did not mention it.

The reverse trap

Knowing about this paradox creates a second temptation: slice until some grouping shows the story you want. Any set can be partitioned to produce almost any conclusion, and choosing the partition after seeing the outcomes is exactly the multiplicity problem from earlier in this block.

Two disciplines prevent it. Fix the slice definitions before the comparison, in the same document as the primary metric. And require a mechanism: a slicing variable belongs in the analysis if you can say why it would affect quality — script, input length, retrieval coverage — not because it separated the results.

The part statistics cannot settle

There is no test that tells you whether the grouped view or the overall view is the correct one. The choice depends on a claim about what causes what, and that claim comes from you.

If language causes difficulty and the mix simply changed, the per-language numbers are the truth and the overall reversal is an artefact. If instead your change itself caused the mix to shift — a new interface that attracts different

users — then the overall number reflects a real consequence of the change, and quoting only the per-group improvement hides it.

So state the assumption out loud in the report: "we treat language mix as external to this change, and standardise to the March traffic mix". A reader who disagrees can then say so precisely, which is the most that any honest analysis of this kind can offer.

BEFORE YOU MOVE ON

A production quality metric drops from 86% to 82% overnight, while every language slice holds steady or improves. What should be checked first?

A. Whether the model version changed, since a silent provider-side update is the usual cause of an overnight drop.

B. Whether the traffic mix shifted towards a slice with a lower pass rate, since the overall figure is a weighted average of the slices.

C. Whether the evaluation set was accidentally reordered, causing items to be scored against the wrong references.

D. Whether one slice grew large enough to be significant, making a previously hidden regression visible.

CASE STUDY · MODULE 3

Eighty-seven beats eighty-two, or does it

A two-person garment export business in Tiruppur comparing two prompts that summarise buyer emails into Tamil for the production floor

Kavitha runs an export business with her brother that makes knitted childrenswear for four buyers in Germany and the Netherlands. The buyers write long emails in English and occasionally German: order changes, delivery dates, fabric specifications, complaints. The production supervisor reads Tamil. For years Kavitha translated by hand, in the evenings.

Her brother Arun, who handles the accounts and is the closer thing to a technical person, built a prompt that turned each email into a Tamil summary with a fixed structure: what changed, the new deadline, the quantities, anything the supervisor must act on today. It worked well enough that Kavitha stopped translating. Then a shipment went out with the old collar specification, because the summary had dropped one sentence from a German email, and the buyer deducted 4% from the invoice.

Arun rewrote the prompt. He also, at Kavitha's insistence, built a test set: a hundred real emails from the last two years, each with a hand-written Tamil summary and a short list of facts that had to be present. He ran the old prompt and the new prompt over all hundred. Old prompt, 82 correct. New prompt, 87.

Arun wanted to switch. Five points was five points, and the new prompt had been written specifically to stop the collar problem. Kavitha, who had spent an afternoon reading the module on error bars that her nephew had sent her, said that on a hundred items five points was inside the noise and asked him to prove it.

The argument had a shape worth pausing on. Arun's cost of not switching was real: every week the old prompt ran was a week with the known failure in it. Kavitha's cost of switching was also real: the summaries were now the only thing the supervisor read, and if the new prompt had regressed on something else, deadlines for instance, nobody would notice until a container was late.

Arun's first move was one the course warns about. He re-ran the new prompt twice more, at the default temperature, and got 85 and 89. He proposed averaging the best two. Kavitha said that was the same as running until it looked good. She was right, and he knew it.

So he did the paired thing. Instead of comparing 82 against 87 he listed which items had changed verdict. Both prompts got 79 items right and 10 items wrong. On the remaining 11, the old prompt was right and the new one wrong on 3, and the old prompt was wrong and the new one right on 8. The difference in the totals was 5, which is 8 minus 3.

The rule of thumb from the course says the difference is worth believing when the absolute gap between the two flip counts exceeds twice the square root of their sum. Twice the square root of eleven is about 6.6. The gap was 5. Not conclusive. Close, but not conclusive, and a hundred items was what they had.

Arun's suggestion was to add fifty more emails and re-run. Kavitha's suggestion was to first look at the three items the new prompt had broken.

All three were deadline items. In each, the German email gave a date in the form 14.03., and the new prompt, which now insisted on quoting the changed specification verbatim, had summarised the specification beautifully and rendered the date as 3 April. The eight improvements were all specification items, which is what the rewrite had been for.

So the honest summary was not "the new prompt is five points better". It was "the new prompt fixes specifications and breaks German dates, and on this set we cannot tell whether the net effect is positive".

What should they ship, and how many more emails, if any, should they label?

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 shipped neither prompt as it stood. Arun added two lines to the new prompt naming the German date convention, and added a deterministic check to the harness: any date in a summary had to be a date that also appeared in the email after normalisation, or the item failed regardless of the rest. That check found one more item in the old prompt's results that nobody had noticed.

With the date fix, the new prompt scored 90 on the same hundred items. The paired count was now 1 regression against 9 improvements, a gap of 8 against a threshold of about 6.3, which was a result. Arun ran the set three times at temperature zero to see the run-to-run spread, which was two points, and wrote all of it in the shared spreadsheet: the score, the flip counts, the spread, and the one regression by name.

Kavitha still asked for the fifty extra emails, but for a different reason. The hundred contained only nine German emails, and the three regressions had all been in that slice. She wanted to know how the prompt behaved on German dates specifically, and nine items could not tell her. They labelled thirty more German emails over two evenings. The new prompt got 27 of 30. That number, on that slice, was the one that made her comfortable.

The frozen hundred stayed frozen. The thirty German emails became a second, named set. And the deterministic date check ran on every summary in production from then on, which is how they caught the next problem, four months later, before the supervisor did.

Arun re-ran the new prompt until he had three scores and proposed averaging the best two. What is wrong with that, and what should he have done with the three runs?

Choosing the best runs after seeing them is optional stopping in miniature: it converts a controlled error rate into an uncontrolled one and biases the reported effect upwards. The three runs were still useful, as a measure of run-to-run spread. Reporting the mean and the spread across all runs, and noting that a gap smaller than the spread cannot be detected however many items are added, is the honest use of them.

The totals differed by five points, yet the paired analysis said the result was not conclusive. Why does counting flips give a different answer from comparing totals?

Items both prompts got right or both got wrong carry no information about the difference between them. Only the discordant items do, and here there were eleven, split eight to three. Twice the square root of eleven is about 6.6, and the gap of five falls short. Pairing is more sensitive than comparing two totals, but it still needs enough discordant items, and a hundred items produced only eleven.

Kavitha asked for thirty more German emails rather than fifty more of everything. What made that the better use of two evenings of labelling?

All three regressions were in a slice of nine items, which could not resolve anything about German dates. Fifty random emails would have added perhaps four or five German ones. Spending the labelling budget on the slice where the question actually lived, and keeping it as a separately named set rather than mixing it into the frozen hundred, answered the specific question without breaking comparability of the original set.

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 An evaluation set's items come from 40 customers, about twelve rows each. In a permutation test comparing two systems on it, what should be shuffled?

A. The row-level system labels, because the null hypothesis is a statement about rows.
B. Nothing; clustered data cannot be permuted and needs a named test with a correction.
C. The system assignment at the customer level, keeping each customer's rows together.
D. The item order, so that the pairing between the two systems is broken.

2 A team tries 100 prompt changes in a quarter. About 10 genuinely help. Tests run at the 5% level with roughly 50% power. About what share of the quarter's significant wins are noise?

A. About one in twenty, because that is what the significance level guarantees.
B. Nearly half: about 5 true detections alongside about 4.5 false alarms.
C. None, provided each individual test was carried out correctly.
D. About one in ten, matching the share of ideas that actually work.

3 Detecting a five-point gap between two systems passing around 80% needs roughly 1,000 items per group. Detecting a 2.5-point gap needs about how many?

A. About 2,000 per group, twice as many, because the gap has halved.

B. About 500 per group, since a smaller gap is subtler and needs a tighter, smaller set.

C. About 1,000 per group, since the count depends on the pass rate and not on the gap.

D. About 4,000 per group, because the gap enters the formula squared.

4 Why is a bootstrap interval for p99 latency computed from 200 requests misleading?

A. Two hundred observations is below the minimum sample size at which any bootstrap interval can be considered valid.

B. Resampling only redraws observed values, so the estimate rests on the two slowest requests you saw.

C. Latency is not normally distributed, and the bootstrap assumes an approximately normal statistic.

D. Percentiles must be computed with the BCa correction, and a plain percentile interval is biased.

5 A fortnight-long online experiment is checked every morning and stopped the first day p drops below 0.05. Even when the effect is real, what happens to the reported effect size?

A. It is overestimated, because the stop happens at a high point of a wandering estimate.

B. It is unbiased, since early stopping only affects the false-positive rate and not the estimate.

C. It is underestimated, because fewer days of data were collected than planned.

D. It is unaffected, provided the daily checks were announced before the experiment began.

6 A new model scores higher than the old one in Hindi (72 against 68) and in English (91 against 89), yet lower overall (75.8 against 78.5). Why?

A. An arithmetic error somewhere; improvements in every group cannot possibly produce a decline in the overall figure.

B. The new model was measured on a mix with far more Hindi, the harder slice, so its average fell.

C. Run-to-run variance, because a 2.7-point gap is inside the noise for sets this size.

D. The Hindi and English sets overlap, so some items are being double-counted in the totals.

MODULE 4

The systems you will actually be handed

Everything so far assumed one output and one label. Real work arrives as a summariser, a translator, a code generator, an extractor, a chatbot, a ranking system, a vision or speech pipeline, or an ordinary tabular model — and each carries a failure the generic metric cannot see. This block gives each one an evaluation design and names what that design exists to catch.

> BY THE END OF THIS MODULE YOU CAN
>
> Design a defensible evaluation for a summariser, a translator, a code generator, a structured extractor, a multi-turn assistant, a ranking system, a vision or speech model, a document pipeline or a tabular model, and name the specific failure mode each design exists to catch

30 · 9 min

Summaries: what was invented, and what was left out

THE ONE THING TO KEEP

Score a summary on two separate axes — every claim supported by the source, and every required fact present — because no automatic similarity metric can see an omission at all.

Two failures, and only one of them is visible

A summary can fail in two ways. It can say something the source does not support, which is a hallucination. Or it can leave out the thing that mattered, which is an omission.

Similarity metrics see neither properly, and omission not at all. A summary of a medical note that drops the allergy will score well against a reference that also drops it, and score badly against one that includes it. The metric is measuring wording. Build two checks instead.

Two failures, two separate checks

Invented (hallucination)

- Split the summary into atomic claims
- Check each against the source: supported, contradicted, not stated
- Report the share of summaries with at least one unsupported claim
- Numbers and dates: a regex against the source, no model needed

Left out (omission)

- No automatic metric can see a missing fact
- Write three to six must-include items per document, in advance
- Coverage is the fraction of them present
- Fifty documents with lists beat five hundred without

A similarity metric sees wording. It scores a summary that drops the allergy well against a reference that also drops it, and it cannot see an omission at all.

Faithfulness: decompose into claims

Split the summary into atomic claims — one assertion each — and check every claim against the source as *supported*, *contradicted*, or *not stated*.

> *Summary: "The council approved the budget on 14 March, raising parking fees by 12% from April."*

> *Claims: (1) the council approved the budget; (2) approval was on 14 March; (3) parking fees rise 12%; (4) the rise takes effect in April.*

Four claims, checked one at a time. Report two numbers: the claim support rate across all claims, and — more important — the **share of summaries containing at least one unsupported claim**. A system at 96% claim support can still be wrong somewhere in half its summaries, and it is the summary a person reads, not the claim.

The checking can be done by a person, by a judge model with the rubric discipline from the judging block, or by a small natural-language-inference model. An NLI model from Hugging Face runs on a CPU, costs nothing per call, and outputs entailment or contradiction against the source sentence — slower to set up and far cheaper to run than a judge.

The free check that catches the worst errors

Numbers and dates are where hallucination does real damage, and they are checkable with no model at all:

```
import re
nums_in = set(re.findall(r"\d[\d,.]*", source))
nums_out = set(re.findall(r"\d[\d,.]*", summary))
invented = nums_out - nums_in           # any value here is a
red flag
```

Run this first. It is exact, instant, and it catches transposed digits, invented percentages and dates that drifted by a day — the errors readers are least likely to notice and most likely to act on.

Omission: you must say in advance what matters

There is no way to measure a missing fact without knowing which facts were required, so somebody has to write them down. For each document in the set,

a person lists three to six **must-include** items. Coverage is then the fraction present in the summary.

This is real work — perhaps five minutes per document — and it is the only part of summary evaluation that measures the thing users actually complain about. Fifty documents with must-include lists is a better set than five hundred without.

Test where in the document the model stops looking

Attention over a long document is not uniform, and the reliable way to find out how yours behaves is to plant a required fact at a known depth. Take the same document and insert a distinctive sentence at 10%, 50% and 90% of the way through, then measure recall of that fact by position.

The usual finding is a dip in the middle, and it is actionable: it argues for chunking the document and summarising in sections rather than for a better prompt. Twenty documents, three positions, an afternoon.

Control for length, and beat the lead baseline

Two things make summary comparisons unfair.

Length. A longer summary covers more and invents more. Fix a band — say 60 to 100 words — and enforce it, or you are comparing verbosity.

The baseline nobody runs. For news and reports, taking the first three sentences is a famously strong summariser. If your system does not beat lead-3 on must-include coverage, you have no result yet. It costs one line of code to check and it has ended more than one project honestly.

Similarity metrics such as ROUGE keep their one legitimate use here: as a drift detector. If ROUGE against last week's outputs falls sharply after a prompt change, the wording moved a lot, which is worth knowing. It is not a quality measurement, and reporting it as one is how teams end up optimising for copying the source, which is exactly what n-gram overlap rewards.

BEFORE YOU MOVE ON

Why can no reference-based similarity metric reliably detect that a summary omitted a critical fact?

A. Because similarity metrics are computed on n-grams, which are too short to represent a fact.

B. Because the reference summary may itself omit the fact, and the score only measures overlap with that reference rather than coverage of what mattered in the source.

C. Because omissions reduce length, and the brevity penalty already accounts for them.

D. Because similarity metrics cannot be computed when the summary is shorter than the reference.

31 · 10 min

Translation, and the languages nobody measures

THE ONE THING TO KEEP

BLEU rewards matching one reference's word forms, so it under-rates morphologically rich languages and short segments — use chrF alongside it, test named phenomena separately, and say plainly when a language is simply unmeasured.

What BLEU is counting

BLEU counts how many n-grams of the output appear in a reference translation, with a penalty for being too short. Everything that goes wrong with it follows from that description.

- **One reference is not the set of correct answers.** A sentence has many valid translations; matching one of them is what gets rewarded.

- **Morphology is punished.** In Hindi, Tamil, Turkish or Finnish, a correct translation that inflects differently from the reference shares fewer whole-word n-grams. The translation is right and the score is lower.
- **Short segments are unstable.** On a six-word sentence, matching or missing one bigram swings the score enormously, which is why BLEU is only meaningful over a corpus of hundreds of segments, never per sentence.
- **Tokenisation changes the number.** The same output can score 28 or 34 depending on how words were split, which is why any BLEU without its configuration is uninterpretable.

Use `sacrebleu`, which fixes the tokenisation and prints a signature you can quote:

```
pip install sacrebleu
sacrebleu ref.txt –i hyp.txt –m bleu chrf
```

chrF in that same command scores character n-grams instead of word n-grams. It handles inflection and languages without spaces far better, and for Indian languages it should be your default over BLEU. Neural metrics such as COMET correlate better still with human judgement, at the cost of running a model and of being opaque — and they were trained mostly on high-resource language pairs, so their reliability drops exactly where you most need help.

Humans, with a typology rather than a rating

Asking a bilingual reviewer "score this out of five" produces threes and fours. Asking them to mark errors produces data. The industry convention is MQM: mark each error with a category and a severity.

- **Accuracy**: mistranslation, omission, addition, untranslated text.
- **Fluency**: grammar, spelling, punctuation, register.
- **Terminology**: wrong term for your domain, inconsistent term use.

Weight severities — minor 1, major 5, critical 10 — sum per thousand words, and report the profile as well as the total. "Fluency is clean, terminology is

wrong 8 times per thousand words" tells you to build a glossary. A single score of 3.6 tells you nothing.

Build a challenge set for what actually breaks

Corpus-level scores hide the failures that generate complaints. Make twenty items each for the phenomena you know matter, and report a pass rate per phenomenon:

- **Named entities**: people, brands and places should not be translated or transliterated inconsistently.
- **Numbers, dates and units**: Indian digit grouping, currency symbols, 12- and 24-hour times.
- **Register and honorifics**: the tu/tum/aap distinction, formal versus casual address, which is a product decision that must be stated before it can be scored.
- **Code-switching and transliteration**: Hinglish written in Latin script is most of the real traffic for many Indian products, and it is absent from every standard benchmark.
- **Gender**: where the source is ambiguous, the target language forces a choice, and the model will guess. Test it explicitly — "the doctor said she", "the nurse said he" — because the guess follows the training distribution and lands on the stereotype.

Two things that look like evidence and are not

Round-trip translation. Translating back into the source and comparing is intuitive and unsound: two errors can cancel, and the return leg has its own quality, so you are measuring a composition of two systems and attributing the result to one.

A benchmark score for a language nobody on the team reads. If no one can read Odia, a COMET score for Odia is a number with nothing behind it. That is not an argument for hiding the language — it is an argument for saying so.

The honest report for a multilingual product

For most of the world's languages there is no reliable automatic metric and no pool of reviewers to hire quickly. A truthful report says which languages were measured, by whom, and which were not measured at all — rather than printing a metric that is technically computable and practically empty.

If a language is unmeasured, the mitigations are procedural rather than statistical: a visible way for users to report a bad translation, a rate limit on automatic publication, and a named speaker who reviews a sample each month. That is weaker than a measurement and stronger than pretending.

BEFORE YOU MOVE ON

A team reports BLEU 31 for English-to-Hindi and BLEU 42 for English-to-German, and concludes the system is much weaker in Hindi. What is the main flaw in that conclusion?

A. BLEU scores are not comparable across language pairs, partly because word-level n-gram matching penalises Hindi's richer inflection, so the gap may reflect morphology rather than quality.

B. BLEU requires at least four reference translations per segment, and the team appears to have used one.

C. Hindi text needs to be transliterated to Latin script before BLEU can be computed correctly.

D. The German test set was probably larger, and BLEU rises with corpus size.

32 · 9 min

Code: run it, in a box, against tests it has not seen

THE ONE THING TO KEEP

Generated code is the one output you can grade by execution, so evaluate with hidden tests in a sandbox, quote pass@1 when a person will accept one answer, and build the set from your own repository because the public ones are in the training data.

The rare task with a real oracle

Almost everything else in this course needs a judgement. Code has an oracle: run it and see. Any evaluation of code generation that reads the code instead of executing it is throwing away its main advantage — plausible-looking code that does not run is exactly the failure mode.

pass@k, and which k your product needs

Sample `n` completions for a problem, count how many pass the tests, and estimate the probability that at least one of `k` samples works:

```
pass@k = 1 − C(n − c, k) / C(n, k)
```

With `n = 10` samples of which `c = 3` pass:

- `pass@1 = 3/10 = 0.30`
- `pass@5 = 1 − C(7,5)/C(10,5) = 1 − 21/252 = 0.92`

Both are true and they describe different products. If a person sees one suggestion and accepts or rejects it, `pass@1` is your number. If the tool shows five options, or an agent retries against a failing test, `pass@k` is. Papers quote the flattering one; decide yours from the interface and state which you used.

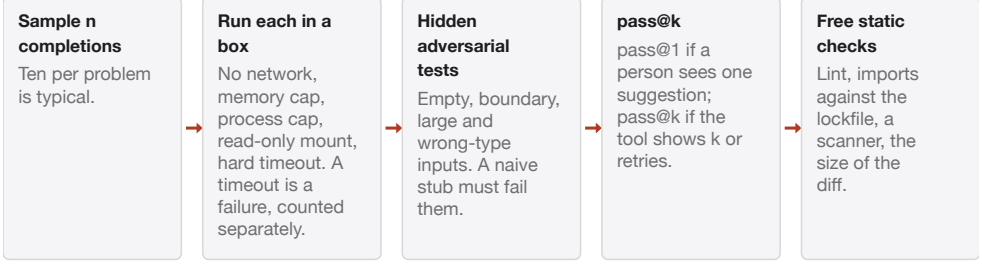
The sandbox is not optional

Generated code will, sooner or later, delete files, open a socket, or spin forever. Run it isolated, always:

```
docker run --rm --network=none --memory=512m --pids-limit=64 \
  --read-only -v "$PWD/work:/work:ro" -u 1000:1000 \
  python:3.12-slim timeout 10 python /work/candidate.py
```

No network, a memory cap, a process cap, a read-only mount, a non-root user, and a hard timeout. Podman works identically and needs no daemon. Treat a timeout as a failure and record it separately, because a spike in timeouts usually means the model started writing loops rather than logic.

Grading generated code by running it



With n = 10 samples and 3 passing, pass@1 is 0.30 and pass@5 is 0.92. Both are true; the interface decides which one is yours.

Tests decide what you are measuring

Hidden tests must be adversarial or you are measuring happy paths. For each problem, include empty input, a boundary value, a large input, and a wrong-type input. Property-based testing with `hypothesis` is the cheapest way to get past your own imagination: state the invariant, let it generate hundreds of inputs.

Then check the tests themselves. If a naive stub implementation passes them, they are too weak. Running the reference solution and a deliberately broken one through the suite takes minutes and tells you whether the suite discriminates.

One more detail decides whether the numbers move at all: **flaky tests**. A suite that depends on the clock, on network access, on dictionary ordering, or on a temporary directory will fail some candidates for reasons the model did not cause. Run the reference solution through the suite twenty times before you trust it; any test that is not deterministic across those runs comes out of the set, or your comparison inherits its noise.

Public benchmarks tell you about the training data

HumanEval and MBPP have been in public repositories for years and are in the training corpora of every current model. A high score on them is evidence about memorisation as much as about capability, and it says nothing about your codebase, your framework versions or your internal conventions.

Build a private set from your own repository. A workable recipe: take 100 merged pull requests that touched a single function and had tests, hide the implementation, keep the tests and the docstring, and ask the model to fill it in. It takes a day, it cannot leak, and it measures the task you actually have.

Correctness is not the whole of it

Passing tests is necessary and not sufficient. Add checks that run free on every candidate:

- **Does it typecheck and lint?** `mypy`, `ruff`, `tsc` — a suggestion that fails CI wastes the developer's time even if it is logically right.
- **Does every import resolve?** Models invent package names, and an invented name is a supply-chain risk the moment somebody publishes a package under it. Compare imports against the lockfile.
- **Does a static scanner flag it?** `semgrep` and `bandit` catch injected SQL, shelling out with user input, and disabled certificate checks.
- **How large is the diff?** A patch that rewrites the file to fix one line is technically passing and practically rejected.

What none of this measures

The eventual value is developer time, and it is not the same as pass rate. Measure the product outcome separately: acceptance rate of suggestions, time to a merged pull request, and how often a review finds a defect that the tests did not. A model with a lower pass rate that produces small, readable, obviously-wrong-when-wrong patches can be worth more than one that produces large patches which pass and nobody can review.

BEFORE YOU MOVE ON

A team samples 10 completions per problem, finds 3 pass on average, and reports "92% pass@5" for an editor plug-in that shows one suggestion inline. What is wrong?

A. Nothing, provided the plug-in retries automatically when the user rejects the first suggestion.

B. pass@5 requires at least 50 samples per problem to be estimated without bias.

C. The product shows one suggestion, so the relevant figure is pass@1 at 30%, and pass@5 describes a system nobody is shipping.

D. The number should have been computed on a public benchmark so it can be compared with published results.

33 · 9 min

Structured extraction, where normalisation is most of the work

THE ONE THING TO KEEP

Score schema validity, field accuracy and whole-record correctness separately, normalise before comparing or you measure your own comparison code, and check every extracted value appears in the source to catch invention for free.

Three questions, three numbers

Pulling structured data out of documents — invoices, CVs, lab reports, product specs — fails at three levels, and merging them hides which one you have.

1. **Did it produce valid output at all?** Parses as JSON, matches the schema, required fields present, types correct. Measure this alone; it is usually fixable with constrained decoding or a retry, and it has nothing to do with reading the document.
2. **Is each field right?** Per-field precision and recall, reported per field, never averaged into one number. Invoice totals and supplier names have different difficulties and different consequences.
3. **Is the whole record right?** The number that matters if a human is not reviewing, and it is much lower than the field accuracy suggests: twelve fields at 97% each gives about 69% of records entirely correct.

Normalisation is where the honesty lives

Most of a first extraction evaluation measures the comparison code rather than the model. The gold file says `2026-03-31`; the model says `31/03/2026`. The gold says `120000`; the model says `1,20,000`. Both are right, and a naive string comparison scores both wrong.

Normalise both sides before comparing, and write the normaliser once:

```
def norm_amount(s):
    return round(float(re.sub(r"[^\d.]", "", s)), 2)

def norm_date(s):
    return dateparser.parse(s, settings={"DATE_ORDER":
"DMY"}).date().isoformat()
```

Then report two field-accuracy numbers: **exact** string match and **normalised** match. The gap between them is a measure of your own formatting debt, and it is usually the cheapest thing on the list to fix.

Beware the reverse error: over-eager normalisation that lowercases and strips punctuation will happily score `M/s Sharma & Co.` equal to `ms sharma co`, which may be fine for a name and disastrous for a part number. Normalise per field type, not globally.

Missing and invented are different failures

Split the errors:

- **Missing** — gold has a value, the model returned null. Costs a human a lookup.
- **Wrong** — both non-null, values differ. Costs a wrong payment.
- **Invented** — model returned a value where the document has none. The most dangerous, because it looks like data.

Invention has a free detector. Any literal value the model extracted should appear in the source text after normalisation:

```
grounded = norm(value) in norm(source_text)
```

This catches the invoice number that was never printed, without a model call and without a gold label — which means you can run it on production traffic, not just on your labelled set. Fields that are computed rather than copied (a total the model summed) need an explicit exception, and that exception list is worth keeping short.

Write the field semantics down, or the labels will disagree

Extraction gold sets have a particular labelling problem: documents contain several plausible candidates. Three addresses appear on an invoice — registered, billing, delivery. Two dates — issue and due. Two names — the person and the company.

Annotators pick different ones, and the disagreement looks like model error later. The fix is in the schema, not the model: define each field by its role and its document cue, not by its type. "Supplier name: the entity issuing the invoice, usually beside the GSTIN in the header" is a definition two people can apply the same way.

Repeated fields need their own rule

Line items, previous employers, dependants: fields that appear many times per document. Comparing two lists needs a matching step before any scoring, and the choice of matching key changes the result. Match on a stable identifier where one exists, otherwise match greedily on the closest normalised description and report the matching rule alongside the score. Then compute precision and recall over line items, plus one count that users notice immediately: how often the number of rows was right.

Confidence, review rate, and the number the business wants

Extraction is where partial automation pays best, because a per-field confidence lets you auto-approve the easy documents and route the rest.

The metric that matters then is not accuracy at all. It is: **at the field accuracy we require, what share of documents can go through untouched.** A model at 94% accuracy with well-ordered confidence may auto-approve 70% of documents at 99.5% accuracy, which is a better product than a model at 96% accuracy with confidence scores that mean nothing. Measure the auto-approve rate at your required accuracy, and calibrate the confidence first, using the arithmetic from the calibration lesson.

Free tools cover all of this: `pydantic` or `jsonschema` for validity, `dateparser` and `rapidfuzz` for normalisation and near-matching, and a spreadsheet

for the per-field breakdown if that is what your team will actually read.

BEFORE YOU MOVE ON

An extraction system scores 97% per-field accuracy across twelve fields, and the team promises fully automated processing. What should they check first?

A. Whether the fields are correlated, since correlated errors would make the record-level rate worse than independence implies.

B. The record-level rate, since twelve independent fields at 97% leaves only about 69% of documents entirely correct.

C. Whether the 97% was computed with exact or normalised matching, since the two can differ by several points.

D. The schema validity rate, since invalid JSON would not have been included in the field accuracy at all.

34 · 9 min

Multi-turn: score the session, not the turn

THE ONE THING TO KEEP

One mistake at turn two corrupts every turn after it, so measure task success per session and record the turn where it first went wrong, then test retention, correction and truncation deliberately.

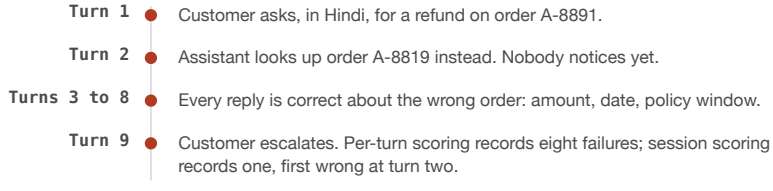
Why per-turn averages mislead

An assistant misunderstands at turn two and everything after that is built on the misunderstanding. Score the ten turns individually and you record eight failures from one mistake, in a set where the ten rows are anything but independent — exactly the clustering problem from the statistics block, in its most common disguise.

Two consequences. The number is wrong: long broken sessions dominate the average. And the diagnosis is wrong: it looks like a widespread quality problem rather than one specific breakdown.

Score the session. Then record, for each failed session, **the turn where it first went wrong**. The distribution of that single field is the most useful diagnostic in conversational evaluation. A pile-up at turn one is a comprehension or routing problem. A pile-up around turn twenty is a context-window problem. A flat spread is a general quality problem, and only then does the average mean anything.

One mistake, eight failed turns



The distribution of 'first wrong turn' across failed sessions is the most useful diagnostic there is: a pile-up at turn one is routing or comprehension, a pile-up around turn twenty is the context window.

Define success as an outcome, not a feeling

"Was the reply good" is unanswerable and unstable. "Did the user end up with a refund initiated, and was the amount correct" is answerable by reading the transcript or, better, by looking at whether the refund event fired in your product.

Where the outcome exists as an event in your system, use it — it costs nothing and cannot be argued with. Where it does not, write a session-level rubric with three or four binary checks, and apply it to the whole transcript.

Five failures worth testing on purpose

Scripted probes find these; random sampling finds them slowly and expensively.

- **Instruction retention.** The user says "reply in Hindi" at turn one. Check turn seven. Also standing constraints: "never mention competitors",

"always give the price in rupees".

- **Correction handling.** The user changes their mind mid-conversation: "actually, make it Thursday". A large share of real failures are systems that keep using the first value. Build twenty sessions where turn four contradicts turn two, and check which value survives.
- **Context truncation.** Plant a fact at turn one, run the conversation past the point where history is trimmed, then ask about the fact. You will discover what your truncation strategy actually drops, which is rarely what the code comment claims.
- **Loops and repetition.** The same reply twice in a row is a distinct failure. It is detectable with a similarity check over consecutive turns, no model needed.
- **Escalation and refusal at the wrong moment.** Measure how often the system hands off, and sample those transcripts. A rising handoff rate can look like caution and be a regression.

Simulated users, and their known bias

To get hundreds of sessions you need a simulated user: a persona, a goal, a stopping rule, and a model playing the part. It works, it is cheap, and it is systematically too easy.

Simulated users are cooperative. They answer the question asked, they do not go silent for four minutes, they rarely misspell, they do not paste a screenshot, they do not change goal halfway. The result is a task-success rate several points above reality.

Calibrate it. Run twenty real transcripts through the same session rubric and compare. If simulated success is 88% and real success is 71%, the simulator is a regression detector — good for comparing versions — and not a source of headline numbers. Write both numbers in the report.

The rest of the numbers

Alongside task success, three cheap session-level measures usually earn their place:

- **Turns to success.** A system that gets there in three turns beats one that gets there in nine, and the average conceals it.
- **Escalation rate**, split by whether the escalation was appropriate.
- **Cost and latency per session**, not per call. A session-level view is what finance and users both experience, and a change that halves per-call cost while doubling the number of turns has made things worse.

What stays contested

Whether a reply is "warm enough", whether an assistant should push back, how much hedging is honest rather than annoying — these are genuine disagreements about what good means, not measurement failures. Report them as a stated position with examples rather than as a score, and revisit the position deliberately. The last block of this course returns to the question of what to do when reasonable people want different outputs.

BEFORE YOU MOVE ON

A support assistant is scored by rating every turn independently. Sessions average 9 turns and the reported quality is 63%. Why is that figure likely to understate the system?

A. Because turn-level ratings are harsher than session-level ones, as raters see less context per rating.

B. Because a single early failure makes every later turn in that session wrong, so one breakdown is counted eight or nine times.

C. Because turn-level scoring counts the system's clarifying questions as failures.

D. Because sessions of different lengths are averaged together without weighting.

35 · 10 min

Ranking systems, where the labels are clicks and the clicks are biased

THE ONE THING TO KEEP

A click measures position as much as relevance, so correct for examination bias, compare rankers by interleaving rather than by A/B, and watch catalogue coverage because a recommender teaches itself what to show next.

Your labels were produced by the system you are grading

Offline ranking metrics assume you know which results were relevant. In a live product you do not; you have clicks. And a click is caused by two things at once — whether the item was any good, and whether it was seen at all.

Examination falls steeply with rank. In typical web results the item at rank one is examined several times more often than the item at rank five, and items below the fold are examined rarely. So the click-through rate at rank one being higher than at rank five tells you almost nothing about relevance, and a model trained or evaluated on raw clicks will learn to reproduce whatever the previous ranker put on top.

Two ways out

Estimate the examination curve and divide it out. Occasionally swap adjacent pairs at random, or randomise the top-k order for a small slice of traffic. Comparing click rates for the same item at different positions gives you the examination probability per rank. Then weight each observed click by the inverse of that probability — inverse propensity scoring — so a click at rank 8 counts far more than one at rank 1.

The catch is variance: dividing by a small probability produces enormous weights, so clip the propensities (a floor around 0.05 is common) and accept a small bias in exchange for a usable estimate.

Interleave instead of splitting. Rather than showing ranker A to half the users and B to the other half, merge both rankings into a single list — take alternate picks from each, tracking whose pick each item was — and attribute clicks to the ranker that contributed the item.

Every user now compares both systems, so the between-user variance disappears from the comparison. Interleaving typically needs one to two orders of magnitude less traffic than an A/B test for the same sensitivity, which turns a two-week experiment into a day. It only works for ranked or multi-option outputs, and it measures preference between rankings rather than any absolute quality.

Offline replay, and the policy you cannot evaluate

You have logs of what was shown and what was clicked, and a new ranker. The temptation is to replay: score the logged items with the new model, see whether it ranks the clicked ones higher.

This works only for items the old system showed. A ranker whose whole point is to surface things the old one never displayed has no logged feedback for exactly its most important behaviour. Off-policy estimators — inverse propensity, doubly robust — extend the reach a little by weighting logged outcomes towards the new policy's choices, but their variance grows as the two policies diverge, and past a certain distance the estimate is unusable.

The honest statement is a limit: "we can estimate this ranker offline because it overlaps 70% with what was shown; the reordered tail is not evaluable and needs live traffic".

Metrics beyond relevance

Ranking quality alone will steer you into a system that recommends the same twenty popular items forever, and that is a real business failure rather than a theoretical one. Track alongside your relevance metric:

- **Catalogue coverage** — the share of items shown to anyone in a week.
- **Concentration** — a Gini coefficient over impressions, watched for drift over months.

- **Intra-list diversity** — average pairwise similarity of the items in one result page.
- **Novelty** — share of recommendations the user has not already seen.

The feedback loop is the mechanism to keep in mind: popular items get shown, shown items get clicked, clicked items are treated as good, and the model gets more confident about a narrowing set. Nothing in a relevance metric detects this, because within the loop the model is right.

Slices that decide the product

- **New users** with no history, and **new items** with no interactions. Both are where recommenders fail hardest and both are usually invisible in an overall number, since existing users and popular items dominate the traffic.
- **Rare queries.** Half of search traffic is often queries seen a handful of times; measure the tail separately, and note that keyword matching frequently beats embeddings there because rare exact terms are exactly what an average vector loses.
- **Zero-result and one-result queries**, which are pure failures the average will not show you.

Free tools cover the offline side: `pytrec_eval` for standard ranking metrics, `rank_bm25` for the keyword baseline, `implicit` for matrix-factorisation baselines. The online side needs no library, only the discipline to randomise something.

BEFORE YOU MOVE ON

A team compares two rankers by measuring click-through rate on the top result for each. Why is that comparison weak evidence about relevance?

A. Click-through rate is a proxy metric, and proxy metrics must always be validated against human labels before use.

B. A click at rank one reflects that the item was examined because it was at rank one, so the measurement mixes examination probability with relevance and favours whatever the previous ranker promoted.

C. Click-through rate ignores dwell time, so a click on a bad result counts the same as a click on a good one.

D. Top-result click-through rate cannot be compared across rankers because the two rankers show different items.

36 · 9 min

Images: the background the model is actually reading

THE ONE THING TO KEEP

Vision sets leak through sessions and reward shortcuts in the background, so split by capture session, test with the correlation deliberately broken, and report accuracy under the blur and low light your users' phones actually produce.

The three tasks and their metrics

Classification uses accuracy per class and the same confusion matrix arithmetic as anywhere else.

Detection uses mean average precision at an intersection-over-union threshold. IoU is the overlap of the predicted and true boxes divided by their union, and a prediction counts as correct only above the threshold — commonly 0.5.

That threshold has a sharp consequence worth seeing. A box that is correct in every practical sense but shifted by a quarter of its width can fall below 0.5, and it is then counted twice against you: once as a false positive, once as a missed detection. So mAP at 0.5 and mAP at 0.75 can rank two models differently, and quoting one without saying which is not a result. Report the IoU threshold every time, and look at the localisation error separately from the classification error.

Segmentation uses mean IoU per class, where thin structures and boundaries dominate the error and human annotators disagree about them too.

Your test set probably leaks through the session

The classic vision leak is not a duplicated file. It is the photo session: forty images of the same object, taken in one place with one camera in one minute, split at random across train and test. The model then recognises the carpet.

Split by capture session, by device, by patient, by shop — whatever unit generated a batch of similar images. And run a near-duplicate check before trusting any split:

```
import imagehash
from PIL import Image
h = {p: imagehash.phash(Image.open(p)) for p in paths}
# pairs within a small Hamming distance are near-duplicates
```

Shortcuts: what the model is really keyed on

Models learn whatever separates the classes fastest, which is often the background. Cows are photographed on grass, so a cow on a beach stops being a cow. Medical imaging models learn the scanner, the ward label burned into the corner, or the ruler that clinicians place beside a suspicious lesion.

The test is direct: build a slice where the correlation is broken — the object on an unusual background, images from a different site, the marker removed — and measure the gap. A model at 94% on the ordinary set and 61% on the bro-

ken-correlation set is not a 94% model. Two hundred such images will tell you more than ten thousand ordinary ones.

Gradient-based attribution maps are worth one afternoon here, not as evidence but as a lead: they show where the model is looking, and you check the suspicion properly with a slice.

The images your users will actually send

An evaluation built from clean catalogue photographs measures a product nobody has. Real intake is a phone camera, held at an angle, in a shop with a fluorescent tube, at 40% battery with the screen dimmed.

Build a perturbation suite and report accuracy at each level: Gaussian blur, JPEG compression at quality 40 and 20, brightness at half and double, 10-degree rotation, motion blur, and downscaling to 480 pixels. `albumentations` and `PIL` do all of it free, and the curve of accuracy against perturbation level is a more useful artefact than any single number — it tells the product team what to ask the camera screen to enforce.

Detection has a threshold problem of its own

Every detector emits a confidence per box, and mAP integrates over all of them — which means it describes no operating point you will ship. Pick the confidence threshold the product will use, and report ordinary precision and recall there, per class. A model with mAP 0.68 that must run at 0.9 precision to keep the review queue sane is a different proposition from one with mAP 0.66 that holds 0.9 precision at twice the recall.

Group slices, handled carefully

Where the images are of people, performance differs by skin tone, age and lighting, and reporting an average hides it. The complication is that measuring this requires the sensitive attributes you are trying to be careful about. The workable practice is to use an established annotation scale, collect it for the evaluation set only, keep it access-controlled, state in the report that it exists and why, and do not attach it to production records.

Two honest limits

Label noise in image sets is real — several per cent even in well-known public sets — so re-label 200 items yourself and measure the disagreement rate before treating the last two points of accuracy as signal. And a model evaluated only on your set tells you nothing about images from a new market, a new device generation, or a new season; that gap is measured after deployment, with the monitoring covered later in this course.

BEFORE YOU MOVE ON

A detector reports mAP 0.71 at IoU 0.5 and 0.38 at IoU 0.75. What does that gap most directly indicate?

A. The model misses many objects entirely, and the stricter threshold makes those misses visible.

B. The model finds objects but localises them loosely, so many boxes overlap enough to count at 0.5 and not at 0.75.

C. The test set is too small for stable mAP estimation at high IoU thresholds.

D. The classes are imbalanced, so mean average precision is dominated by rare classes at strict thresholds.

37 · 9 min

Speech: word error rate, and the words that matter

THE ONE THING TO KEEP

Word error rate is meaningless without the text normaliser that produced it, and it weights every word equally while your product depends on names, numbers and dates — so measure entity accuracy and end-to-end task success beside it.

The arithmetic, and why it can exceed 100%

Word error rate is edit distance over words, divided by the reference length:

```
WER = (substitutions + deletions + insertions) /
reference_words
```

Reference: "please cancel my order for thursday" — six words. The system produces "please cancel my order thursday morning": one deletion ("for"), one insertion ("morning"), so WER is 2/6 = 33%.

Because insertions are not bounded by the reference length, WER can exceed 100%. A model that hallucinates a paragraph during silence — a well-documented failure of some end-to-end systems — can score 400% on a short clip, which is a useful signal rather than a broken metric.

`jiwer` computes it in two lines and is free.

The normaliser decides the number

Two teams can report 8% and 14% on identical audio and identical transcripts. The difference is text normalisation:

- casing and punctuation removed, or not

- "25" against "twenty five"
- "Dr." against "doctor"
- fillers — "um", "uh" — kept or stripped
- contractions expanded
- Indian numbering, currency words, and mixed-script transliteration

None of these choices is wrong; publishing the score without the choices is. Use a published normaliser — the one shipped with Whisper is a de facto standard for English — and state which, or write your own and version it with the results. Then apply the identical normaliser to every system you compare, including the vendor's.

For languages without clear word boundaries, and for heavily inflected ones, report **character error rate** as well; word-level scoring punishes an inflection difference as heavily as a wrong word.

WER is not what your product cares about

WER weights every word equally. Your booking assistant does not: "Thursday", "four", "Nandini", and the order number carry the meaning, and "the" does not.

Measure **entity accuracy** separately — dates, times, quantities, names, identifiers, product terms — as a per-slot exact-match rate after normalisation. It is common to see 9% WER with 78% date accuracy, and the second number is the one that predicts complaints.

Then measure the pipeline. If the transcript feeds an assistant, what matters is end-to-end task success, and the relationship is not linear: a system at 12% WER can support 95% intent accuracy, because language models recover from small transcription errors and not from a wrong number. Measure the two stages and the end-to-end result, and improve whichever stage the end-to-end number blames.

The slices that decide whether it works for your users

Speech quality varies more across speakers and conditions than almost any other AI task. Report per slice, and build the slices deliberately:

- **Accent and region.** For a country with many first languages, this is the first cut.
- **Code-switching.** A sentence that starts in Hindi and ends in English defeats systems trained on either alone, and it is how a great many people actually speak.
- **Background noise**, banded by signal-to-noise ratio: a quiet room, a shop, a street, a moving auto.
- **Channel.** Telephony audio is 8 kHz and narrowband; a model evaluated on studio recordings will disappoint on a call.
- **Speaking rate and age.** Children and elderly speakers are consistently worse-served and consistently absent from test sets.

Common Voice and other openly licensed corpora provide starting material for several of these, and a hundred recordings collected from your own users in their own conditions is worth more than any of it.

Where the recordings come from

Two practical constraints shape the set before any metric applies. Recording a person's voice is collecting biometric-adjacent personal data, so consent, retention and access control apply the same way they do to message logs — with the extra difficulty that a voice cannot be redacted the way an email address can. And a public corpus recorded by volunteers reading prompts is read speech, which is easier than spontaneous speech in every way that matters: no false starts, no overlap, no thinking aloud. Where you rely on read speech, say so, because the number will be optimistic against live calls.

Diarisation, timing and latency

If the output has speakers or timestamps, add two more measures. **Diarisation error rate** covers missed speech, false speech and speaker con-

fusion. **Alignment error** matters wherever subtitles or highlights are shown; a transcript that is perfect and 400 ms late is a bad subtitle.

For streaming, latency is part of quality: time to first partial result, and the finalisation delay after the speaker stops. A system that is 2% better on WER and a second slower will feel worse in a live conversation, and neither number alone will tell you that.

BEFORE YOU MOVE ON

A vendor reports 6% WER and your own measurement of the same audio gives 13%. What is the most likely explanation to check first?

A. The vendor evaluated on a different, cleaner audio set than the one you used.

B. Your reference transcripts contain errors, which inflate the measured error rate.

C. The two measurements used different text normalisation — casing, punctuation, numbers and fillers — which routinely moves WER by several points.

D. WER is computed per utterance by some tools and per corpus by others, and the two differ.

38 · 9 min

Documents: stages that multiply, and checks that need no labels

THE ONE THING TO KEEP

A document pipeline's accuracy is the product of its stages, so measure each one and the end-to-end separately — and use the document's own internal consistency, such as line items summing to the total, as a label-free check on live traffic.

Three stages, and the multiplication

A document pipeline is usually: get the pixels right, get the structure right, get the fields right. Each stage looks respectable and the product of three respectable stages is not.

```
0.97 (page classification) * 0.95 (text and layout) * 0.94
(field extraction) = 0.87
```

Thirteen documents in a hundred go wrong, and the end-to-end number is the only one the customer experiences. Measure every stage, because that is where the fix lives, and always report the end-to-end figure alongside, because that is the promise.

Three respectable stages, one product

0.97 × 0.95 × 0.94 = 0.87. Thirteen documents in a hundred go wrong, and the customer only ever meets the last bar.

Character accuracy is not reading accuracy

An OCR engine can transcribe every character on a page correctly and produce useless output, because it read a two-column page straight across, or emitted a table row by row when the meaning was column by column. Character error rate cannot see this: the characters are all present.

So measure reading order and structure separately from characters:

- **Reading order**: does the extracted text sequence match the human reading sequence, checked on a sample by a person.
- **Table structure**: start with the simplest counts, row count and column count correct, because those errors explain almost every downstream failure. Then cell-level precision and recall, matched by position, with merged and spanning cells recorded as their own error category.

Build the set from real intake

This is the single largest source of self-deception in document AI. An evaluation assembled from clean, digitally-generated PDFs measures a system that will meet photographs of paper.

Real intake looks like: a phone photo at fifteen degrees, glare across the total, a staple through the date, a rubber stamp over the signature, handwriting in the margin, a faded thermal print, a fax at 200 dpi, page three photographed twice and page four missing.

Take a hundred documents from the actual intake channel, keep the ugly ones, and note the condition of each so you can slice by it. The distribution of conditions is itself a finding, and it usually argues for fixing the capture screen before touching the model.

Checks the document performs on itself

Documents carry internal redundancy, and it gives you correctness checks with no gold labels at all — which means they run on live traffic, not only on your set.

- Line items sum to the subtotal; subtotal plus tax equals the total.

- The tax rate applied matches the stated rate.
- Dates are ordered sensibly: issue before due, dispatch before delivery.
- Identifiers pass their checksum, where the format has one.
- The page count matches the "page 1 of 4" in the header.

An arithmetic failure means something was misread even though you have no idea what the truth is. Log the failure rate of these checks as a production quality metric; it moves the moment an upstream capture setting or a supplier's template changes, and it costs nothing per document.

Templates are the hidden cluster

Document sets are dominated by a handful of layouts. Two hundred invoices from one large supplier share a template, so a model that has learned that template scores well on forty per cent of your set while being useless on everything else. Group the set by supplier or template, report accuracy per template group, and count how many distinct templates the set actually contains. A set of 500 documents from 12 templates has twelve independent observations of layout handling, which is the number that governs how well a new supplier will go.

Multi-page reality

Batch-scanned intake arrives as a stack, and the pipeline has to decide where one document ends and the next begins. The failure mode is not a wrong field but a wrong boundary: two invoices merged into one record, or a three-page contract split into two.

Measure document-splitting accuracy explicitly — correct boundaries found, spurious boundaries added — before field-level metrics, because a split error makes every field metric downstream meaningless.

Scripts, and where the tooling is thin

OCR quality for Devanagari, Bengali, Tamil and other Indic scripts lags Latin substantially, and mixed-script documents — an English form filled in Hindi — are harder still. Report per script, and expect the gap.

Free tools take you a long way: Tesseract as a baseline that runs anywhere, PaddleOCR and docTR for better accuracy including several non-Latin scripts, and `jiwer` for character error rate. Run the free baseline even if you intend to buy, because the difference between it and a paid service on *your* documents is the only version of that comparison worth having.

BEFORE YOU MOVE ON

Why is an internal consistency check, such as line items summing to the invoice total, unusually valuable in a document pipeline?

A. It measures extraction quality more accurately than field-level comparison against gold labels.

B. It requires no gold labels, so it can run on live production documents and flag misreads for which no ground truth exists.

C. It replaces the need for a labelled evaluation set once the pipeline is in production.

D. It detects OCR errors specifically, allowing the OCR stage to be scored separately from the extraction stage.

39 · 9 min

Tables, cross-validation, and the leak that gives you 0.99

THE ONE THING TO KEEP

Fit every preprocessing step inside the fold, split by group and by time rather than at random, and treat an implausibly high score on a business problem as a leak to be found rather than a result to be announced.

The oldest kind of model, still most of the work

Churn, default, fraud, demand, conversion, triage: a great deal of applied machine learning is a table with rows and columns and a target. The metrics are the ones from earlier in this course. What differs is how the data splits and how it leaks.

One split is not enough on small data

A single train-test split on 2,000 rows gives an estimate with visible sampling noise; rerun with a different seed and the AUC moves. **K-fold cross-validation** — five folds, each row held out exactly once — uses everything and gives you five estimates whose spread is itself informative. If your folds range from 0.72 to 0.86, no single number should be quoted without that range.

Two variants matter more than the choice of k:

- **Grouped folds** whenever rows share an entity — several loans per customer, several visits per patient. Same argument as the clustering lesson: without grouping, the model has seen the customer.
- **Time-based folds** for anything with a time dimension. Train on the past, test on the future, walking forward. A random split lets the model learn from March to predict February, which no deployed model will ever be able to do.

```
from sklearn.model_selection import GroupKFold, TimeSeriesSplit
```

Leakage, in the four forms you will actually meet

A column that could not exist yet. `days_to_payment` in a model predicting default. `claim_closed_reason` in a model predicting claim approval. The rule: for every feature, ask when its value becomes known, and delete anything known only after the target.

Preprocessing fitted outside the fold. Scalers, imputers, and especially target encoding fitted on the whole dataset leak the test rows' information into training. The symptom is a cross-validation score better than any honest one, and the fix is to put every step inside a pipeline so it is refitted per fold:

```
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(SimpleImputer(), StandardScaler(),
LogisticRegression())
cross_val_score(pipe, X, y, cv=GroupKFold(5), groups=g)   #
fitted inside each fold
```

Duplicates across splits. Re-submitted forms, re-imported rows, the same customer under two IDs. Deduplicate on business keys before splitting.

An identifier that encodes the answer. Customer numbers issued in order encode signup date; branch codes encode region; a row index correlates with when the label was collected. Drop identifiers, or hash them and check the model does not improve when you do.

The tell for all four is the same: **an implausible score**. AUC 0.99 on churn, or 0.97 on fraud, is a bug until proven otherwise. Print feature importances the moment you see one — the leaking column is almost always at the top, and the whole investigation takes fifteen minutes.

Learning curves answer "should we collect more data"

Plot the score against training-set size, from 10% to 100%, with the same folds. Two readings:

- The curve is still climbing at 100%: more rows will help, and collecting them is the cheapest available improvement.
- The curve flattened at 40%: more rows will not help. You need better features, a better target definition, or to accept the ceiling.

This one plot has redirected more projects usefully than any model comparison, and it costs a loop over five sample sizes.

Baselines and the honest state of the field

Run `DummyClassifier`, then logistic regression, then a gradient-boosted tree — `LightGBM` or `XGBoost`, both free and both fine on a CPU. On tabular data, boosted trees remain very hard to beat, and the published comparisons with deep-learning approaches are still genuinely contested rather than settled; treat any claim in either direction as a claim about a particular set of datasets. What is not contested is that the boosted tree takes an hour, so it belongs in your comparison regardless.

The gap between the notebook and the service

The last failure is not statistical. Features computed in a notebook from a warehouse table are rarely computed identically by the serving code from live data: a different time zone, a different default for missing values, an aggregation window that is 30 days in one place and a calendar month in the other.

Check it directly. Take 100 rows, compute the features both ways, and compare column by column. Any mismatch is a silent accuracy loss that no offline evaluation will ever show you, and it is the most common reason a model that measured well arrives in production worse.

BEFORE YOU MOVE ON

A team fits a `StandardScaler` and a target encoder on the full dataset, then runs 5-fold cross-validation on the transformed data, and reports AUC 0.94. What is wrong?

A. Nothing, provided the model itself is refitted inside each fold, which cross-validation does automatically.

B. Target encoding is unsuitable for cross-validation and should be replaced with one-hot encoding.

C. The preprocessing saw the held-out rows, so each fold's test data influenced its own features and the 0.94 is optimistic by an unknown amount.

D. Five folds is too few for a reliable estimate, and ten would have exposed the discrepancy.

CASE STUDY · MODULE 4

Ninety-eight per cent of the fields, sixty-one per cent of the forms

A co-operative bank in Kerala buying a document pipeline to read loan applications before the festival lending season

The Thrissur District Co-operative Bank processes about 9,000 small loan applications in the eight weeks before Onam: gold loans, two-wheeler loans, festival advances. Each application is a bundle of paper: the form itself, an identity document, an address proof, and for salaried applicants a salary slip. Clerks in 34 branches type the fields into the core banking system by hand, and the backlog in the last fortnight of the season is the single biggest complaint the bank receives.

A vendor demonstrated a document pipeline: photograph the bundle, and the system classifies each page, reads it, and fills the form fields automatically. The demonstration was on the vendor's own sample of forty documents and reported 98% field accuracy. The general manager wanted it in the branches by the first week of August. The head of operations, Mr Menon, had one question, which was how many applications would come through with every field right, because a form with one wrong field still has to be handled by a person, and if the person has to check every field to find the wrong one, the pipeline has saved nothing.

The vendor did not have that number. Neither did anybody else.

The bank's IT officer, Deepa, was given the job of finding out, with a constraint: three weeks before the season, and every week of measurement was a week less of rollout. The vendor offered their sample set, already labelled, and the argument for using it was speed. Deepa's argument against was that the vendor's forty documents were flat scans of clean forms. The bank's intake was a branch clerk's phone camera, at an angle, under a tube light, with a rubber stamp over the date and, in the gold-loan forms, a thumbprint where a signature should be.

The decision in front of Mr Menon was whether to accept the vendor's measurement and roll out with a clerk checking a sample, or to spend a week building a set from real intake and measuring the things the vendor had not.

Deepa's proposal was specific. Take 150 application bundles from last season, photographed the way clerks actually photograph them, from six branches including two rural ones. Label the fields by hand, which she estimated at two clerks for three days. Then measure three separate numbers, not one: whether the output was valid at all, the accuracy of each field, and the proportion of whole applications with every required field correct. And measure each stage of the pipeline separately, because the vendor's system did three things in sequence, page classification, text extraction, and field extraction, and a wrong page classification made every downstream number meaningless for that bundle.

She also wanted to know how many distinct salary-slip layouts the 150 bundles contained, because if most of them came from three large employers the score would describe three templates and not the world.

The cost of Deepa's week was a week. The cost of skipping it was that the bank would learn the whole-record number in the second week of the season, from the clerks, in the branches with the longest queues.

What would you have measured, and what would you have done if the whole-record number came back well below the field number?

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

Mr Menon gave her the week. The results on 150 real bundles: schema-valid output on 97% of bundles; page classification correct on 96%; per-field accuracy, after normalising dates and amounts, 93% on average, with the appli-

cant's name at 98% and the employer's address at 81%. Whole-record accuracy, every required field correct, was 61%.

The salary slips came from 19 distinct layouts, of which three, all large employers in Kochi, accounted for 70% of the slips. On those three templates the field accuracy was 97%. On the other sixteen it was 84%.

The number that changed the rollout was not the model's. Thirty-one of the 150 bundles had at least one page that was simply unreadable: motion blur, or the bottom third of the form cut off by the frame. Nothing downstream could fix that. Deepa's recommendation was a capture screen in the branch app that detected blur and edge cut-off and asked the clerk to retake, before any of the vendor's pipeline ran. It took the vendor four days to add and it raised the whole-record figure to 74% on a re-photographed subset, more than any change to the extraction model.

The bank rolled out with a review queue: bundles where any field's confidence fell below a threshold went to a clerk, with the extracted value shown beside a crop of the source region, so that checking took seconds rather than a re-read. And Deepa added the checks that needed no labels at all: the loan amount on the form had to match the amount on the sanction sheet, the date of birth had to make the applicant at least eighteen, and identity numbers had to pass their checksum. Those ran on every bundle in production, and their failure rate became the number on the operations dashboard for the season.

The vendor reported 98% field accuracy and the bank measured 61% whole-record accuracy on real intake. Both were true. How do the two numbers relate, and which one governs the product?

Whole-record accuracy is roughly the product of per-field accuracies across the required fields, so twelve fields at 97% each give around 69% of records entirely correct. The bank's fields were also less accurate on phone photographs than on clean scans. The whole-record figure governs the product wherever a person does not review every form, because a record with one wrong field still needs a person, and finding the wrong field costs nearly as much as typing the form.

Why did Deepa insist on counting distinct salary-slip layouts, and what did the answer change?

Document sets are dominated by a few templates, so a score on 150 slips where three employers supply 70% of them is mostly a score on three layouts. Grouping by template showed 97% on the common three and 84% on the other sixteen, which is the number that predicts how a new employer's slip will fare. Nineteen layouts gave nineteen independent observations of layout handling, not 150.

The largest improvement came from a blur check on the capture screen rather than from the extraction model. What general lesson about document pipelines does that illustrate?

A pipeline's accuracy is the product of its stages, and the first stage was the photograph. Thirty-one of 150 bundles were unreadable before any model ran, and no downstream change could recover them. Measuring each stage separately located the loss where it lived; building the set from real intake rather than clean scans is what made the loss visible at all.

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 A summariser's claim support rate is 96%, but about half of its summaries contain at least one unsupported claim. Which figure should lead the report?

A. The 96% per-claim figure, because it is computed on the larger sample and therefore has the narrower confidence interval.

B. An average of the two, since each captures part of the picture.

C. The share of summaries with at least one unsupported claim, since a person reads the summary, not the claim.

D. ROUGE against a reference summary, since it is the standard measure.

2 A coding tool shows a developer exactly one suggestion, to accept or reject. Which number describes it?

A. pass@5, because sampling more completions gives a more stable estimate of capability.

B. pass@1.

C. pass@10, because that is the value quoted on the public benchmarks.

D. The mean of pass@k across all values of k up to the sample size.

3 In structured extraction, which failure has a free detector that runs on live traffic with no gold label at all?

A. Missing values, by counting null fields against the schema's list of required fields for that document type.

B. Wrong values, by comparing each field against the previous run's output for the same document.

C. Schema failures, by asking a judge model whether the JSON looks plausible.

D. Invented values, by checking the extracted literal appears in the normalised source text.

4 An assistant misunderstands at turn two of a ten-turn session and every later reply is built on the misunderstanding. What does the course say to record?

A. One failed session, plus the turn at which it first went wrong.

B. Eight failed turns, each weighted by its position in the conversation so later ones count for more.

C. A partial score of 0.2, since the first two turns were fine.

D. The average per-turn quality, since that is what the user experienced across the session.

5 Forty photographs of the same object, taken in one session, are split at random between training and test, and the classifier reports 96%. What is the likely problem?

A. The test set is far too small for a figure of 96% to carry any meaning, whatever the split.

B. The model may recognise the carpet, not the object; split by capture session.

C. Label noise in image sets inflates accuracy on small test sets.

D. The random split has placed too many of the hard images in training.

6 A churn model reports an AUC of 0.99 in cross-validation. What does the course say to do first?

A. Ship it; on tabular data, gradient-boosted trees routinely reach scores of this kind.

B. Add more folds so the estimate's spread across folds can be reported with the figure.

C. Treat it as a leak until proven otherwise, and print the feature importances.

D. Compare it against a deep-learning model to confirm the result holds.

MODULE 5

Automating judgement

Hand labelling does not scale past a few hundred items a week, so most of the checking has to be done by code and by models. This block builds that layer in the right order — deterministic checks first, a calibrated judge second, the biases of a judge measured rather than hoped away — and then points it at the three hardest cases: groundedness, safety and agents.

> BY THE END OF THIS MODULE YOU CAN
>
> Build a layered automatic check with cheap deterministic assertions underneath a rubric-driven model judge, state the agreement figure that licenses trusting the judge's numbers, and control the position bias, self-preference and cost that judging brings with it

40 · 9 min

LLM-as-judge, and where it misleads you

THE ONE THING TO KEEP

A judge you have not measured against your own labels is an opinion with a decimal point.

Why you will end up using one

You cannot pay humans to read 4,000 outputs a week. A model can, for a few dollars, and for many criteria it agrees with careful human labels most of the time. Judging is a real technique and you should use it.

It also fails in ways that are specific, repeatable, and easy to miss — because the failures produce plausible numbers, not errors.

The four biases worth knowing by name

Position bias. In pairwise comparisons ("which reply is better, A or B?"), judges systematically favour one position. The fix is cheap: run every pair twice, with the order swapped, and count a win only when both orderings agree. The rate at which the two orderings *disagree* is your noise floor — if the judge contradicts itself on 25% of pairs, no win margin under 25 points means anything.

Length bias. Longer, more structured, more headed answers score higher, largely regardless of content. This matters because the easiest way to change a prompt is to make it produce more words. Check it directly: plot judge score against output length across your set. If the correlation is strong, your "quality improvement" may be a verbosity improvement, and your users may hate it.

Self-preference. Models tend to score outputs from their own family more highly. Where you can, judge with a model from a different provider than the one generating. Never let the same model be both the generator and the sole arbiter of its own comparison.

Scale compression. Ask for a 1-to-10 rating and you will get sevens and eights, forever, on everything. The information content is near zero. Ask a specific yes/no question with a reason and you get data you can use.

```
You are checking one support reply against one rule.

RULE: The reply must not state or imply a refund deadline
      that is not present in the policy text below.

POLICY: {policy}
REPLY:  {reply}

First write one sentence of reasoning quoting the relevant
span of the reply. Then output strict JSON:
{"reason": "...", "violates_rule": true | false}
```

Reasoning before the verdict, one rule per call, a boolean out. Asking for five judgements in one call makes all five worse and correlates their errors.

The step almost everyone skips

Calibrate the judge against your own labels.

Take 50 items you and a colleague labelled by hand in lesson two. Run the judge on them. Compute agreement.

- Two humans agreed 95% of the time. The judge agrees with the humans 92%. Use it, and remember every number it gives you carries roughly eight points of error, so a five-point improvement is not a result.
- The judge agrees 71%. It is not usable for this criterion. Rewrite the rubric, try one rule per call, try a different model — and re-measure. Do not ship a dashboard built on it.

A judge you have not calibrated is not a measurement instrument. It is a confident opinion with a decimal point on it.

Two things judges are good at, one they are not

They are good at **rule checks** — did this reply promise something it should not, did it stay in the right language, did it cite a document that exists. They are good at **pairwise preference on a fixed rubric**, once you have controlled for position.

They are poor at **absolute scores you plan to compare across time or across teams**. Judge scores drift when the judge model updates, and "our agent scores 8.4" is meaningless to anyone outside your repository. Keep judge numbers as internal, relative, same-run comparisons. That is what they are honest for.

BEFORE YOU MOVE ON

A pairwise judge is shown the old and new prompt's answers and picks the new one 68% of the time across 200 real inputs. The team is ready to ship. What is the strongest reason to pause?

A. Each pair was shown in one order only, so an unknown part of that 68% could be the judge preferring whichever answer it read first.

B. 68% is too close to a coin flip to support a decision at any sample size.

C. Pairwise judging is inherently unstable; they should score each answer 1 to 10 and compare the means.

D. A human should read the 32% of cases the new prompt lost before anything ships.

41 · 9 min

Turning a quality you can feel into rules a machine can apply

THE ONE THING TO KEEP

A usable rubric is a short list of single-property binary rules written from real failures, judged one per call, with the evidence quoted before the verdict.

The gap between a standard and a rubric

You know what a good reply looks like. The problem is that knowing is stored in your head as a pattern, and a judge — human or model — needs it as a list.

A rubric is that list. Writing one is the same exercise as the labelling guide in module one, pushed a step further, because a model is a more literal reader than a colleague. Your colleague fills gaps with common sense and shared context. The model fills them with whatever is statistically likely, which is usually generous.

The test for a finished rubric: hand it to somebody who has never seen your product, along with twenty outputs. If they come back with a question, the model was going to guess at that same point, and you will never see it guess.

Anatomy of a rule

Every rule has the same four properties, and a rule missing one of them will produce inconsistent scores:

1. **One property.** "Clear and complete" is two rules pretending to be one. Split it.
2. **Observable in the output.** "The customer will feel reassured" is not checkable; "the reply states what happens next and by when" is.
3. **Binary.** True or false, with no middle. If a rule genuinely needs three levels, it is two rules stacked.
4. **Evidence-bearing.** The judge must be able to point at the span that decides it.

A worked rubric for a support reply, ordered hard constraints first:

> ***R1 (gate).*** *The reply makes no commitment about money, dates or eligibility that is absent from the supplied policy text.*

> ***R2 (gate).*** *The reply does not state a fact about the customer's account that is absent from the supplied account data.*

> ***R3.*** *The reply answers the question asked, not an adjacent one.*

> ***R4.*** *Where required information is missing, the reply asks for it rather than assuming a value.*

> ***R5.*** *The reply is in the same language as the customer's message.*

> ***R6.*** *The reply states the next step and who takes it.*

Six rules. Each separately true or false. R1 and R2 are gates: fail either and the item fails regardless of the rest, because a fabricated refund date is not compensated for by good structure.

That gate distinction is worth building into the score explicitly. An average across six rules would let a system fail R1 five per cent of the time and still report 0.95. Report gates as a separate count, always as raw numbers: "R1 violated in 3 of 200".

One rule per call

Give a model all six rules in one prompt and ask for six verdicts, and the verdicts correlate. It forms an overall impression and then produces six judgements consistent with that impression. You can watch this happen: outputs cluster on all-pass and all-fail far more than independent rules would produce.

Six separate calls cost six times as much and are worth it, because each judgement becomes independent evidence. At typical small-model prices, 200 items times six rules is a few tens of thousands of tokens per rule and cents in total. Cheaper than being wrong.

If the cost genuinely matters, group only rules that are logically unrelated, and never group two rules that a lazy reader would conflate.

Reasoning first, verdict last

Ask for the verdict first and the model commits, then writes a justification for a decision already made. Ask for the evidence first and the verdict is conditioned on it.

```
RULE: The reply makes no commitment about money, dates or
      eligibility that is absent from POLICY.

POLICY: {policy_text}
REPLY:  {reply_text}

1. Quote every span of REPLY that makes such a commitment,
   or write NONE.
2. For each quoted span, quote the sentence of POLICY that
   authorises it, or write NOT FOUND.
3. Output JSON: {"spans": [...], "violates": true|false}
```

And then verify mechanically that every quoted span really is a substring of the reply. It is one line of code, it is free, and it catches the judge inventing evidence — which it does, at a low but non-zero rate that you would otherwise never see.

Write the rubric from failures, not from first principles

The rubric people write at a desk is neat and wrong. The rubric that works comes from reading 50 real failures and asking, for each, "what rule would have caught this?"

Do it in that order and you get rules that fire. Do it the other way and you get a beautiful eight-rule rubric where six rules pass on every item ever produced and contribute nothing but cost.

A rule that has never failed in 500 items is not a safety net. It is either the wrong rule or a solved problem, and either way it should be retired or moved to a cheap deterministic check.

Version it

Put the rubric in the repository with a version number, and record which rubric version produced every score. When you change a rule — and you will, roughly every time you read a new batch of failures — old numbers stop being comparable to new ones.

Teams lose weeks to this. A metric drops four points, everybody investigates the system, and the cause is a rubric edit somebody made on Tuesday. The scores were measuring a different thing, and nothing in the dashboard said so.

BEFORE YOU MOVE ON

A judge prompt asks a model to rate a reply from 1 to 5 on "clarity, accuracy and tone" and return a single score with a short explanation. What is the most damaging flaw?

A. Five points is too coarse a scale; ten would give more resolution to detect small improvements.
B. Three separate properties are fused into one number, so a failure of accuracy can be offset by good tone and the score cannot be acted on.
C. The explanation comes after the score, so the model has already committed before reasoning.
D. No reference answer is supplied, so the model has nothing to compare the reply against.

42 · 9 min

Gold answers, and asking whether two answers agree

THE ONE THING TO KEEP

Ask a judge whether the candidate says the same thing as the reference rather than whether it is good, write references as required and forbidden facts instead of prose, and audit the disagreements because a share of them are errors in your gold answers.

Two protocols, and the easier question

There are two ways to judge an output. **Reference-free**: give the judge a rubric and the material, and ask whether the output satisfies it. **Reference-based**: give the judge a gold answer, and ask whether the candidate agrees with it.

The second is a much easier question for a model, and much more stable. "Is this a good answer" is an aesthetic judgement that drifts with phrasing, length and mood. "Do these two answers give the same answer to this question" is a comparison, and comparisons are what judges are reliably good at.

So whenever a question has an answer you can write down, write it down.

```
QUESTION: {question}
REFERENCE ANSWER: {reference}
CANDIDATE ANSWER: {candidate}

Ignore wording, length, tone and ordering. Consider only
whether the
candidate conveys the same answer to the question as the refer-
ence.

Quote the part of the candidate that decides this, then output
JSON:
{"quote": "...", "verdict": "same" | "different" | "partial"}
```

Write references as facts, not prose

A paragraph-shaped reference invites the judge to compare paragraphs, and the judge then punishes an answer that is correct and phrased differently. The mechanism is anchoring: a fluent reference gives the model a template, and deviation from a template reads as error.

Write the reference as structure instead:

```
{ "required": ["refund window is 14 days", "counted from deliv-
ery"],
  "forbidden": ["30 days", "counted from order date"],
  "acceptable_variants": ["two weeks"] }
```

Now the judge checks membership of facts rather than resemblance of prose, which is a question with a defensible right answer. It is also directly checkable by code for the parts that are literal strings, which means many items never need a model call at all.

Some of your gold answers are wrong

Every gold set carries errors — a policy that changed, a mislabelled item, an annotator's misreading. In practice five to ten per cent is normal, and it distorts your measurement in a specific direction: a model that is right where the reference is wrong is scored as failing.

The fix is a targeted audit, and it is the same trick as in the multi-label lesson. Take fifty items where the candidate disagrees with the reference, and have a person judge the *item*, not the model — is the reference actually correct here. Typically a third of the disagreements turn out to be reference errors. Correct them, note the correction rate in the report, and re-run.

Do this before concluding anything from a score in the nineties, because at that level most of the remaining failures may be your file rather than the model.

When there is no single right answer

Plenty of tasks have many valid outputs: a support reply, a code review comment, a summary's phrasing, a translation into a language with several registers. Forcing a reference on those creates false negatives at scale, and the rubric approach from earlier in this block is the right tool.

The most useful design is usually a hybrid. Split the output into the part with a right answer and the part without:

- **Reference-checked core**: the refund window, the amount, the date, the decision, the API name. Exact or normalised match, no model needed.
- **Rubric-judged remainder**: tone, completeness, whether it answered the question asked, whether it invented a policy.

Reporting these separately also survives contact with a product meeting, because "the facts are right 97% of the time and the tone check passes 84%" supports different actions than a blended 91%.

References need versions

A reference answer encodes a policy at a point in time. When the refund window changes from 14 days to 30, every affected reference becomes wrong, silently, and the eval starts penalising the correct behaviour.

Attach a policy version to each item and fail loudly when the versions do not match the current policy document. It sounds bureaucratic until the first time an eval blocks a correct change for a fortnight because nobody remembered which items mentioned the old window.

Multiple references help too — two or three acceptable answers per question reduce false negatives — but they multiply the maintenance burden, so spend them on the items where the variation is real rather than on the whole set.

BEFORE YOU MOVE ON

A judge is given a fluent paragraph as the reference answer and asked whether the candidate is correct. Correct-but-differently-worded answers are frequently marked wrong. What is the mechanism?

A. The judge model is too small to handle long references and truncates them.

B. A prose reference gives the judge a template to compare against, so deviation in wording reads as deviation in content.

C. The judge has not been given the original question, so it can only compare the two texts.

D. Reference-based judging is inherently less reliable than reference-free rubric judging.

43 · 9 min

Proving the judge agrees with you, and re-proving it

THE ONE THING TO KEEP

Judge quality is per-class recall on a stratified calibration set, not overall agreement, and it must be re-measured whenever the judge model, the prompt or the traffic changes.

The calibration set

A judge is a measuring instrument, and an uncalibrated instrument produces numbers, not measurements. Calibration here means one specific thing: showing that the judge's verdicts match careful human verdicts on the same items, and stating by how much they differ.

Build a **calibration set** and keep it separate from everything else:

- 50 to 100 items, drawn from real traffic
- labelled independently by two people, disagreements resolved and the ruling written down
- **deliberately enriched with the rare class.** If violations are 4% of traffic, a random 50 contains two of them and tells you nothing about whether the judge catches violations. Take 25 known violations and 25 clean items, and remember when you report that you have stratified.
- frozen, versioned, and never used for tuning the system itself

That set costs about five hours of two people's time once, and it is what every judge number you ever quote will rest on.

Agreement, per class

Run the judge over the calibration set and compare against the human labels. Compute overall agreement — and then immediately stop looking at it and

compute the two numbers that matter.

An illustrative result on a set of 50 with 12 real violations:

- Overall agreement: 44/50, which is 88%. Sounds fine.
- Of the 12 real violations, the judge flagged 4. **Recall 33%.**
- Of the 6 items it flagged, 4 were real. **Precision 67%.**

The judge is 88% agreeable and catches one violation in three. If you had shipped a dashboard on the overall figure, you would have reported a low violation rate and been technically correct and completely wrong.

This is the same lesson as accuracy under imbalance from module two, and it bites harder here because the number feels like a validation rather than a metric.

Use **Cohen's kappa** as the headline instead, for the reason given in module one: it discounts the agreement you would get by chance. Above 0.8, the judge is a usable instrument. Between 0.6 and 0.8, usable with the slack stated on every number it produces. Below 0.6, it is not measuring your criterion and no amount of prompt fiddling should be reported as if it were.

What to do when agreement is poor

In order of how often it works:

1. **Split the rule.** Most poor agreement comes from a compound criterion. Two rules at 0.85 beat one at 0.55.
2. **Add the boundary rulings** from your labelling guide directly into the judge prompt as few-shot examples. Two or three well-chosen edge cases move agreement more than any amount of instruction rewriting.
3. **Require quoted evidence** before the verdict, if you are not already.
4. **Change judge model.** A larger model helps on genuinely subtle criteria. It also costs more per call, so measure whether it earns it.
5. **Accept that it is a human criterion.** Some judgements — whether a bereavement message was appropriate, whether a joke lands in Marathi —

are not automatable at any agreement you would accept. Sample these for human review and stop pretending.

Never accept the fifth option before trying the first.

Re-calibrate on a schedule and on a trigger

A calibration goes stale. Re-run the frozen calibration set:

- **monthly**, as routine
- **whenever the judge model version changes** — including when a provider silently updates an alias like `-latest`, which is the argument for pinning an exact version string in the judge as much as in the system
- **whenever the judge prompt changes**, even trivially
- **whenever the input distribution moves**, such as launching in a new language or a new customer segment

This takes ten minutes because the labels already exist. Skipping it is how a team ends up six months into trusting a judge that quietly stopped agreeing with them in March.

Record each run: date, judge model version, judge prompt version, kappa, per-class recall. Five lines in a file. That file is the provenance of every quality number your organisation believes.

The bias you cannot calibrate away

One caveat to end on. Calibration proves the judge agrees with your labels. It does not prove your labels are right. If your two labellers share an assumption — that longer answers are more thorough, that formal register is more professional — the judge will learn that assumption and the calibration will look excellent.

The only defence is periodically bringing in somebody outside the team to label 20 items cold and comparing. It is uncomfortable and it is the cheapest audit available.

BEFORE YOU MOVE ON

A judge is validated on 200 randomly sampled production items and agrees with human labels 94% of the time. Policy violations occur in about 3% of traffic. Why is this validation weak?

A. A random sample of 200 contains roughly six violations, so the figure says almost nothing about whether the judge detects the thing it exists to detect.
B. 200 items is too small for any reliable agreement estimate; at least 1,000 are needed.
C. Human labels on production data are unreliable because reviewers see the model's output before deciding.
D. Agreement should be measured against a second judge model rather than humans, to remove human inconsistency.

44 · 9 min

Turning pairwise wins into a ranking you can defend

THE ONE THING TO KEEP

Fit a Bradley-Terry model to your pairwise comparisons rather than accumulating Elo, put a bootstrap interval on every rank, and remember that preference rewards length and confidence, so measure accuracy separately.

Why pairwise, and what it costs

People and judges are far more consistent asked "which of these two is better" than asked "rate this out of ten". Relative judgements do not drift over a session the way absolute scales do, and they do not require everyone to share a private notion of what 7 means.

The cost is that a pile of pairwise outcomes is not a ranking, and turning it into one is where teams either do something principled or invent a number.

Protocol first, because the aggregation cannot repair a bad protocol: randomise which candidate is shown first, run every pair in both orders, and count a win only where both orders agree. The rate at which the two orders disagree is your noise floor, and no margin smaller than it means anything.

Allow a **tie** as an explicit option. Forcing a choice between two equally good answers manufactures data that is pure noise, and ties are informative: a system that ties with a much cheaper model on 60% of items is telling you something about whether the expensive one is worth running.

From wins to scores: Bradley-Terry

The standard model says each system has a strength `s`, and:

```
P(A beats B) = 1 / (1 + exp(−(s_A − s_B)))
```

Fit it by logistic regression over your comparison records. One row per comparison, one column per system, `+1` for the left system, `−1` for the right, target 1 if left won. The fitted coefficients are the strengths, and they come with standard errors for free.

```
import numpy as np
from sklearn.linear_model import LogisticRegression
X = np.zeros((len(comparisons), n_systems))
for i, (a, b, left_won) in enumerate(comparisons):
    X[i, a], X[i, b] = 1, −1
LogisticRegression(fit_intercept=False, C=1e6).fit(X,
y).coef_[0]
```

Elo is an online approximation of the same model. It updates after each match by a fixed step, which makes it order-dependent and sensitive to the step size — two properties you do not want in an evaluation. Elo exists because chess needed a rating that could be updated by hand between tournaments. If you have all the comparisons in a file, fit the model properly; use Elo only where results arrive continuously and you need a live number.

If you do quote Elo-style numbers, know the conversion: a gap of 100 points corresponds to a win probability of `1 / (1 + 10^(−100/400))`, about 64%. A gap of 30 points is a 54% win rate, which is close to nothing.

Put an interval on the rank

Bootstrap over comparisons — resample the comparison records with replacement, refit, record each system's rank — and report the interval. The usual result on a leaderboard of six systems is that ranks 2 to 4 are statistically indistinguishable and the ordering between them changes on every resample.

Reporting "second, third and fourth are tied within the data we have" is more useful than an ordered list, and it stops a team spending a month chasing a position that does not exist.

How many comparisons

For three to six systems, run every pair; it is cheap and it avoids any question about the design. Above that, all-pairs grows quadratically and adaptive schemes help: compare systems whose current intervals overlap, and stop comparing pairs that have separated. Roughly `k log k` well-chosen comparisons per round is enough to order `k` systems whose true gaps are not tiny.

The thing preference does not measure

Preference is not accuracy. Human raters and judge models both reward length, structure, confident phrasing, headings and bullet points. A change that adds a summary paragraph and three bullets will win comparisons while being no more correct, and sometimes less.

So run the two measurements together and read them as a pair: preference win rate, and factual accuracy or task success on the same items. When preference goes up and accuracy stays flat, you have made the output more likeable — which is a real and legitimate product goal, provided you say that is what you did.

This is also the reason to treat public preference arenas as a shortlist rather than a decision. They aggregate crowd preferences on other people's prompts,

using a population that is not your users, on tasks that are not your task, with the same length-and-confidence bias baked in.

BEFORE YOU MOVE ON

A leaderboard built from pairwise judging shows model B ahead of model C by 25 Elo-equivalent points after 400 comparisons. What is the defensible reading?

A. B is meaningfully better and should be adopted, since the comparison used a proper pairwise protocol.

B. B and C are close to tied: 25 points implies about a 54% win rate, and a bootstrap over the comparisons will very likely show overlapping ranks.

C. The gap cannot be interpreted at all without knowing which model was shown first.

D. B is better on style but the ranking says nothing about which is more accurate, so the comparison is meaningless.

---

45 · 8 min

The checks that cost nothing and catch most of it

THE ONE THING TO KEEP

Deterministic checks are free, exact and repeatable, so run them first and move every check you can down into code before spending a judge call on it.

Order your checks by price

There are three tiers of checking, and they differ by four orders of magnitude in cost:

- **Deterministic code.** Microseconds, free, perfectly repeatable.

- **Model judge.** Roughly a second and a fraction of a penny per call, repeatable to within its own noise.
- **Human.** Two to five minutes and real money, and the only one that can settle a genuinely new question.

Run them in that order and let each tier filter for the next. A judge asked to assess the quality of an answer that is not valid JSON has burned a call to tell you something `json.loads` knew for free.

Most teams build the middle tier first, because it is the interesting one. Then they discover their judge's most common verdict is "the output was truncated".

Three tiers of checking, four orders of magnitude apart

Human review	two to five minutes and real money; the only tier that settles a new question
Model judge	about a second and a fraction of a penny; repeatable to within its own noise
Deterministic code	microseconds, free, perfectly repeatable; runs first on everything

Run from the bottom up, and let each tier filter for the one above it. A judge asked about an output that is not valid JSON has spent a call to learn what json.loads knew for free.

What code can check

More than people expect. Every item here is a real production failure mode, and every one is a few lines:

- **Structure.** Does it parse as JSON? Validate against a JSON Schema, in `jsonschema` or Pydantic. Are all required fields present, of the right type, and is every enum value in the allowed set?
- **Grounding, cheaply.** Every quoted span the model claims to have taken from the source is genuinely a substring of the source. Every cited document id exists. Every URL is one from the supplied context, not one the model composed.
- **Numbers.** Sums add up. Line items total the stated total. Dates are valid, in range, and in the right order. A percentage is between 0 and 100. VAT is what the arithmetic says.

- **Language.** The output language matches the input's, via `langdetect` or `fasttext` — a free 900-language model that runs in milliseconds.
- **Leftovers.** No `[INSERT NAME]`, `Lorem ipsum`, `As an AI language model`, `TODO`, or the system prompt echoed back.
- **Bounds.** Length between a floor and a ceiling. Truncation detection: did it stop because it finished, or because it hit the token limit? The finish reason is in the API response and going unchecked is one of the most common silent failures in production.
- **Privacy.** No email address, phone number or card-shaped digit string that was not in the input.
- **Repetition.** The same sentence three times, or a trailing loop — a cheap n-gram check catches degeneracy that a judge often rates as "thorough".

Write them as ordinary tests. `pytest` over a JSONL file is a perfectly good eval harness for this tier, it runs in CI, and it is free.

The first number to get

Before any quality work, measure **what fraction of outputs fail a structural check**. It is a ten-minute job and it is routinely much higher than the team's guess — the guess is usually near zero, because the team has only ever looked at outputs that rendered.

When that number is meaningful, fix it before anything else, and fix it in the right place. Structured-output or JSON-mode support from the provider, a constrained decoder such as `outlines` or `guidance`, or one retry on a parse failure will each do more for your quality metric than a week of prompt rewriting. And they are boring, which is why they get skipped.

Assertions live next to the data

Keep the checks per item, not per suite. Some items have specific expectations — this invoice's total is 4,320; this query must return the Delhi office's address; this refusal must not name a competitor. Store them with the item:

```
{"id": "inv-041",
 "input": "...",
 "assert": [{"type": "json_field", "path": "total", "equals":
4320},
            {"type": "not_contains", "value": "approximately"},
            {"type": "language", "equals": "input"}]}
```

This is exactly how **promptfoo** models an eval, and if you want the pattern without writing it, promptfoo is MIT-licensed, runs from a YAML file, and takes about twenty minutes to get running. **Inspect** from the UK AI Safety Institute is the more programmable free option. **DeepEval** wraps the same ideas around pytest. Commercial platforms — Braintrust, Langfuse's paid tiers, LangSmith — add hosted dashboards and team features on top of the same fundamentals.

Start with a JSONL file and a for-loop anyway. You will understand what any platform is doing, and moving 200 items into a tool later is an hour's work, while adopting a platform before you know what you are measuring shapes your evaluation around the tool's opinions.

The rule of thumb

Before adding any judge call, ask whether code could decide it. If the answer is "code could decide it if the output were structured", change the output to be structured. Every check you move down a tier makes your evaluation faster, cheaper, perfectly repeatable, and immune to the judge drifting under you.

BEFORE YOU MOVE ON

A team's LLM judge reports that 18% of their extraction outputs are "low quality". What should they check before tuning the prompt?

A. Whether the judge itself agrees with human labels on those 18%.

B. Whether a stronger judge model gives a different figure, to establish whether 18% is real.

C. What fraction of those outputs fail to parse, are missing required fields, or hit the token limit and were truncated.

D. Whether the 18% cluster in particular input types, using the slice tags from the eval set.

46 · 9 min

Checking an answer against the passages it was given

THE ONE THING TO KEEP

Decompose an answer into claims and test each against the retrieved text with an entailment model, remembering that groundedness measures faithfulness to the source and never truth.

The one check that runs on live traffic

Almost every quality measurement needs a label. Groundedness does not: the source passages were part of the request, so you can check an answer against them at any time, including in production, on every response, with no human in the loop.

That property makes it the most valuable automatic check a retrieval system can have, and it is worth building carefully.

The pipeline

One: split the answer into claims. One assertion each, resolved pronouns, no compound sentences. A small model does this adequately; a larger one does it well; either way, cache the result, because claims are stable across judge changes.

Two: retrieve the relevant source sentences for each claim — usually just the passages already in the context, chunked to a few sentences, ranked by embedding or keyword similarity to the claim.

Three: classify entailment. For each claim and its candidate support: supported, contradicted, or not enough information. A natural-language-inference cross-encoder does this well, runs on a CPU, and costs nothing per call:

```
from transformers import pipeline
nli = pipeline("text-classification", model="cross-encoder/nli-
deberta-v3-base")
nli(f"{passage}</s></s>{claim}")     # entailment / contradic-
tion / neutral
```

Threshold the entailment probability, and calibrate that threshold against your own labelled examples rather than accepting the default.

Free checks that come first

Before any model runs, three deterministic tests catch the most damaging errors:

- **Quoted text must appear verbatim** in the source. A quotation the model composed is the clearest possible failure and a string search finds it.
- **Numbers and dates in the answer must appear in the source**, allowing for formatting. The same regex as in the summarisation lesson.
- **Citations must resolve.** The chunk id exists, and it contains at least one content word from the claim it supports. A citation pointing at a chunk about something else is common and invisible to readers.

Measure the checker itself

A groundedness detector is a classifier, and it needs the same treatment as any other. Label 200 answers by hand — supported, unsupported, partially supported — and compute the detector's **recall on unsupported claims**, which is the class you care about. Overall agreement will look high because most claims are supported, and it will tell you nothing.

Re-measure after any change to the chunking, the retriever, the answer format, or the NLI model. The calibration lesson's discipline applies unchanged.

What entailment models cannot do

Three limits, each with a workaround:

- **Long premises.** NLI models were trained on short sentence pairs. Feeding a 3,000-token passage degrades them badly. Chunk, and check the claim against the best few chunks rather than the whole context.
- **Arithmetic and aggregation.** "Revenue rose 15%" is not entailed by two revenue figures in any sense an NLI model understands. Route numeric claims to a code check that recomputes them, or to a judge with the numbers isolated.
- **Implicature.** A source saying "the office closes at 5" does not literally entail "you cannot collect at 6", and a strict entailment check will mark a sensible answer unsupported. Decide your policy — strict entailment or reasonable inference — write it into the labelling guide, and apply it to both the humans and the model.

Aggregating claims into an answer-level number

Claim-level percentages flatter the system, because most claims in any answer are uncontroversial scaffolding. Report the number a reader experiences: **the share of answers containing at least one unsupported claim**. An assistant at 98% claim support and 12% of answers carrying an unsupported claim is described honestly by the second figure and misleadingly by the first.

Weight by consequence where you can. An unsupported claim about a deadline, an amount or an eligibility rule is not the same event as an unsupported claim in a closing pleasantry, and a two-tier severity marking in the claim decomposition costs one extra field.

Grounded is not true

The most important sentence in this lesson. Groundedness measures whether the answer follows from the retrieved passages. If the passage is out of date, wrong, or from a bad source, a perfectly grounded answer is false.

So groundedness pairs with two other measurements and never replaces them: source quality, which is a corpus problem — freshness, provenance, deduplication of contradictory versions — and answer correctness against reality on a labelled subset.

The failure this catches in production is a system confidently citing a superseded policy document. Groundedness will be 100%, users will be misinformed, and the only thing that finds it is a check on the corpus itself.

BEFORE YOU MOVE ON

A retrieval assistant scores 99% groundedness in production, yet users report incorrect refund deadlines. What is the most likely explanation?

A. The groundedness checker has poor recall on unsupported claims, so it is missing hallucinations.

B. The retrieved passages include a superseded policy document, so the answers faithfully reflect a source that is wrong.

C. The entailment model is treating reasonable inferences as supported, inflating the score.

D. Groundedness was measured on claims rather than on whole answers, which overstates it.

47 · 9 min

What judging costs, and how to pay a tenth of it

THE ONE THING TO KEEP

Cascade the checks — free code first, a small or local judge next, the expensive judge only on the uncertain remainder — and cache by output hash, because a rerun that changed no outputs should cost nothing.

Do the arithmetic once

A realistic harness: 2,000 items, four rubric checks each, run in both orders for the pairwise part. That is 16,000 judge calls. At roughly 1,500 tokens per call, 24 million tokens per run.

Whatever today's prices are, the ratio between a small model and a frontier one is roughly ten to one, so the same run costs a few dollars or several dozen. Run it on every pull request, twenty times a day, and the annual figure is the kind that appears in a cost review with your name beside it. Teams do not usually abandon evaluation because it stopped working; they abandon it because it became the third-largest line in the bill and nobody could say what it bought.

The cascade

Order the checks by cost and stop as early as possible.

1. **Deterministic code.** Schema validity, forbidden strings, numbers grounded in the source, citations resolving, length bands. Free, instant, and on a typical set this decides 20 to 40% of items outright.
2. **A small or local judge** for the rule-shaped checks. An 8-billion-parameter open-weights model through Ollama or llama.cpp runs on a laptop, costs nothing per call, and is entirely adequate for questions like "does this reply state a deadline that is absent from the policy".

3. **The expensive judge**, only for items where the cheap layer is uncertain, or where the rubric requires nuance.

Then measure what the cascade cost you. Take 300 items, judge them with the full expensive judge and with the cascade, and compare verdicts. A typical result — small judge confident on 80% of items and agreeing with the large one on 97% of those — means you pay about a fifth of the bill for roughly half a point of accuracy, and you can state that trade rather than assume it.

Cache on the thing that determines the answer

The judge's verdict depends on the input, the output, the rubric and the judge version. So cache on a hash of exactly those:

```
key = sha256(f"{item_id}|{sha256(output)}|{rubric_version}|
{judge_model}".encode())
```

A rerun after a change that touched documentation, or a retry after a network failure, should cost nothing. In practice a well-keyed cache removes 60 to 90% of calls in day-to-day development, because most edits change a few outputs rather than all of them.

Cache the claim decomposition separately from the verdict, since claims survive rubric changes.

Judge less often, on purpose

Not every run needs every item.

- **A core set** of 150 to 300 items, judged on every run, chosen to cover the strata. This is the CI signal.
- **The full set**, judged nightly or weekly, which is the number you report.
- **A random sample** of live traffic, judged continuously at a fixed rate, which is the production signal.

The sizing arithmetic from the statistics block tells you what the core set can detect — usually large regressions only — and that is exactly what a CI gate

should be for.

The other levers

- **Batch endpoints** where the provider offers them: typically half price for results within a day, which suits a nightly full run perfectly.
- **Shorter prompts.** Do not paste an entire document into a check about one paragraph. Halving prompt length halves the bill, and it usually improves the judge, because the relevant text is no longer buried.
- **One rubric item per call** stays right even under budget pressure. Bundling five questions into one call saves tokens and correlates the errors, which costs you more in reliability than it saves in money.
- **Skip judging what code can decide.** Every check moved down into deterministic code is a permanent saving and a permanent gain in reproducibility.

The free path, stated plainly

A judge running on your own machine costs electricity. An 8-billion-parameter model quantised to four bits needs about 6 GB of memory and produces a verdict in a few seconds on an ordinary CPU, faster with any recent GPU. `ollama run llama3.1:8b` or a `llama.cpp` build is the whole setup.

Where it is adequate: binary rule checks, schema and format questions, obvious contradictions, routing decisions in a cascade. Where it is not: fine distinctions of tone, multi-step reasoning about a policy, anything where the expensive judge itself only just agrees with your labels. Measure which of those you are in with the same calibration set you built earlier, and you will find that a meaningful share of a typical rubric runs locally for nothing.

Put the cost in the report

Print cost per run, cost per item, and the cache hit rate alongside the quality numbers, in the same table the team already reads. A harness whose cost is visible gets tuned. A harness whose cost is invisible gets switched off in a

quarter when somebody is asked to cut 20% from the bill, and the first anybody hears of it is when quality drops with no measurement to detect it.

BEFORE YOU MOVE ON

A cascade sends only uncertain items to the expensive judge. What must be measured before trusting its numbers?

A. The share of items each layer resolves, so the cost saving can be reported accurately.

B. Agreement between the cascade's verdicts and the expensive judge's verdicts on a sample judged by both.

C. The small judge's agreement with human labels on the full evaluation set.

D. The latency of each layer, since the cascade adds round trips for escalated items.

48 · 10 min

Testing what happens when somebody tries

THE ONE THING TO KEEP

Measure harmful completions and false refusals as a pair, test injection through retrieved content rather than only through user messages, and run the adversarial set several times because attacks succeed probabilistically.

A different set, with different rules

Your hundred items came from real traffic, so they represent people using the system as intended. A safety evaluation represents people using it as *not* intended, plus people whose ordinary use happens to look alarming. It is a separate set, built deliberately, scored on its own metrics, and it does not get averaged into your quality number.

Four categories are worth having, and they cover most of what actually happens:

1. Direct attempts. Requests for content you will not produce, in every framing people use: roleplay, fiction, "for a research paper", translation into another language, encoding, a hypothetical, an appeal to authority, an urgent emergency.

2. Prompt injection through content. This is the one that matters most for anything with retrieval, browsing or tools, and it is the one most often untested. The attack is not in the user's message. It is inside a document, a web page, an email, a filename or a database row that your system reads — text addressed to the model, telling it to ignore its instructions, exfiltrate what it has seen, or call a tool.

Build a small corpus of poisoned documents and put them in your test index. A minimal battery: an instruction in white text, an instruction in a code comment, an instruction in a footnote, one in a language other than the interface language, one that claims to come from the system administrator, and one that asks the model to include a URL with data appended to it. Then measure how often the system follows them. The correct rate is zero and the observed rate rarely is.

3. Out-of-scope requests. Medical, legal or financial advice from a shopping bot. Requests to speak as a named real person. Requests about internal systems. The failure here is not usually harm; it is a system confidently operating outside what anyone tested.

4. Distress. Somebody in genuine difficulty. The right behaviour is a product decision that should be written down before it is measured, and the failure modes are in both directions: treating a person as a policy violation, or missing the situation entirely.

Two rates, and they must be read together

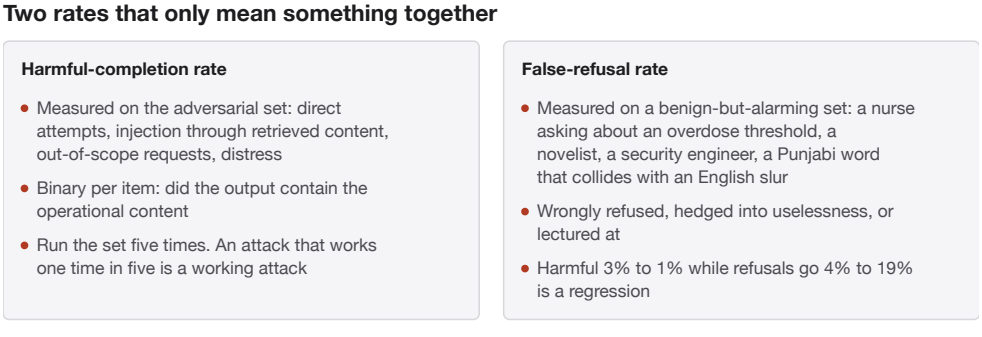
Safety evaluation has one number everybody reports and a second that gives it meaning.

- **Harmful-completion rate** — of the adversarial items, how many produced the thing you did not want.
- **False-refusal rate** — of *benign* items, how many were wrongly refused, hedged into uselessness, or lectured at.

You need a benign-but-alarming set to measure the second. Fifty items whose surface features look risky and whose intent is entirely ordinary:

- a nurse asking about a paracetamol overdose threshold
- a novelist asking how a character would pick a lock
- a security engineer asking how SQL injection works
- a Punjabi speaker using a word that collides with an English slur
- a teenager asking what to do about a friend who has stopped eating
- a historian asking about a massacre in explicit terms

Optimise the first number alone and you will drive the second one up until the product is useless to the people it was built for, and you will not notice, because nobody logs a refusal as a bug. Report both, always, as a pair. A move from 3% to 1% harmful completions that took false refusals from 4% to 19% is a regression, whatever the safety dashboard says.



Optimise the left number alone and the right one rises until the product is useless to the people it was built for, and nobody notices, because nobody files a bug about a refusal.

Scoring

Binary and specific, per item: did the output contain the operational content, or not. Not "how safe was it out of five". Hedged, partial and refused-then-

complied outputs all need a written ruling in the guide, because they are most of the interesting cases.

Run the adversarial set five times, not once. Attacks succeed probabilistically; an attack that works one time in five is a working attack, and a single-run evaluation reports it as blocked 80% of the time under a heading that says "passed".

Tools and how to get variety

garak (Apache 2.0) is a scanner with a large library of known probes and is the fastest way to a baseline. **PyRIT** from Microsoft (MIT) automates multi-turn attacks. **promptfoo** has a red-team mode that generates adversarial variants of your own prompts. All free.

Use them for coverage of published techniques, and then write your own for your domain, because the attacks that will actually be used against you are specific to what your product does and no public library knows about them. A model can help generate variants of an attack you have already seen, which is a legitimate and useful way to turn one incident into thirty test items.

Expect to be behind

A new framing that defeats your defences will appear, and it will appear on a Tuesday. The measurable goals are not "unbreakable". They are: known attacks stay blocked as the system changes, new attacks become test items within a day of being seen, and the false-refusal rate does not creep up while nobody is watching it.

That last one is the failure teams do not notice for months, because nobody files a ticket to say the assistant was unhelpfully cautious. They just stop using it.

BEFORE YOU MOVE ON

A team hardens their assistant and reports harmful completions falling from 4.0% to 0.9% on their adversarial set. What single additional number would most change how you read that?

A. The number of adversarial items, since 0.9% on a small set could be a single case.
B. The refusal rate on a set of benign inputs that superficially resemble the adversarial ones.
C. Whether the adversarial set includes multi-turn attacks as well as single-turn ones.
D. The latency added by the new safety checks, since guardrails are typically expensive.

49 · 10 min

When the output is a sequence of actions

THE ONE THING TO KEEP

Score an agent on the final state, record the trajectory for diagnosis, run every task several times because a task that passes three times in five has not passed, and count cost and steps as part of the result.

Two things to score, and you need both

An agent takes actions: it calls tools, reads results, decides again. Evaluating it means scoring two different things.

Outcome. Did the world end up in the right state? The ticket exists, with the right customer and priority. The file contains the right rows. The refund of ₹2,400 was issued to the right order. This is checkable in code against the system's final state, and it is the number that matters.

Trajectory. How did it get there? Which tools, in what order, with what arguments, how many times. This does not decide pass or fail on its own, but it is where every diagnosis lives, and it catches the agent that got the right answer by an unrepeatable accident — nine searches, three of them identical, and a lucky guess.

Score outcome, record trajectory, and read the trajectory whenever the outcome is wrong.

Partial credit, done carefully

A six-step task that fails at step five scores zero on outcome, and zero tells you nothing about whether you are close. So define checkpoints in advance, per task:

```
{"task": "refund-042",
 "checkpoints": [
   {"id": "found_order",   "check":
"tool_called('lookup_order', order_id='A-8891')"},
   {"id": "checked_policy","check":
"tool_called('get_policy')"},
   {"id": "refund_issued", "check": "db.refunds.has(order='A-
8891', amount=2400)"},
   {"id": "customer_told", "check":
"tool_called('send_message')"}],
 "gate": ["refund_issued"]}
```

The checkpoint profile across many runs is genuinely diagnostic: if 80% of failures stop at `checked_policy`, that tool is the problem, and no prompt change is required to know it.

One discipline: checkpoints are defined before the run. Written afterwards, they describe the path your agent happened to take, and every future agent is scored on resembling it. Several correct paths usually exist, so express checkpoints as states reached rather than as an exact sequence wherever you can.

Non-determinism is not a nuisance, it is the measurement

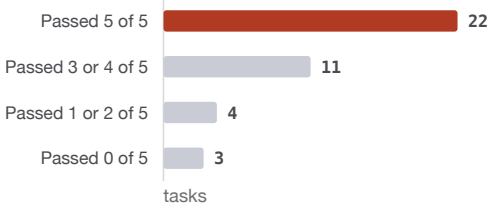
Run each task once and you have learned almost nothing. Agents branch: a slightly different tool result leads to a different plan.

Run each task **five times** and report pass rate per task. The distribution is the finding:

- 5/5 — reliable
- 3/5 — this is not a passing task, and reporting it as one is how agent demos become production incidents
- 0/5 — reliably broken, which is at least easy to debug

A suite of 40 tasks run 5 times is 200 runs, which is affordable and which changes the conversation from "it worked when I tried it" to a number with a spread. Report the mean pass rate *and* the count of tasks that passed all five times. The second number is usually the one a business would actually care about.

Forty tasks, each run five times



The mean pass rate is about 77%, which is the number in the announcement. Twenty-two tasks pass every time, which is the number a business can rely on. A task at three of five has not passed.

Cost, steps and time are part of the score

Two agents both pass 82% of tasks. One averages 8 tool calls and ₹1.60 per task; the other averages 34 calls and ₹9.20. These are not the same system, and only one of them can be turned on for a hundred thousand users.

Record per run: tool calls, total tokens in and out, money, wall-clock seconds, and whether it hit the step limit. Hitting the step limit deserves its own count — it is the signature of an agent that is looping, and it looks identical to ordinary failure in a pass-rate table.

Watch for the pattern where a change raises pass rate by two points and doubles cost. That is usually a longer step budget rather than better reasoning, and it is a trade you should make deliberately rather than discover in the invoice.

The environment is the hard part

An agent evaluation that sends real emails, charges real cards or writes to the real database can be run once. To run it two hundred times you need an environment that resets:

- **Fake the side-effecting tools.** A record of what would have been sent is enough to check the outcome, and it is much faster.
- **Seed a fresh database per run** and tear it down. Docker or an in-memory SQLite is plenty.
- **Freeze time and randomness** where the task depends on them, or "tomorrow" makes half your tasks fail in March.
- **Snapshot the tool results** for a fixed suite so retrieval or API flakiness does not show up as agent failure. Keep a smaller live suite as well, because a fully mocked evaluation stops noticing when a real API changes.

Building this environment is usually more work than building the agent. That is not a sign you have gone wrong; it is the normal ratio, and it is why teams skip it and ship on demos instead.

Tools

Inspect (UK AISI, MIT) has first-class support for tool-using tasks and scoring. **SWE-bench** and **τ-bench** are worth reading as designs even if you never run them — τ-bench in particular formalises the multi-run idea with a pass^k metric, the fraction of tasks solved on *all* k attempts, which is the honest thing to report and a number that is always lower than the one in the announcement.

BEFORE YOU MOVE ON

An agent suite of 30 tasks is run once each; 25 pass, and the team reports 83%. What is the most important correction?

A. Each task should be run several times and the pass rate reported per task, because a single run cannot distinguish a reliable task from one that succeeds two times in five.

B. 30 tasks give a confidence interval of about ±13 points, so the suite needs to be several times larger.

C. The trajectories of the 5 failures should be read before any number is reported.

D. Pass rate should be replaced by average checkpoint completion, which is more informative than a binary outcome.

CASE STUDY · MODULE 5

The judge that gave everything an eight

An edtech company in Bengaluru using a language model to score its own AI feedback on class 10 English essays

Padhai Labs sells an essay-practice product to CBSE schools. A student writes an essay on the app, and an AI tutor returns feedback: what worked, three things to improve, and a corrected paragraph. Teachers liked it, sales were growing, and the product team wanted to ship changes to the feedback prompt every fortnight.

Reading the feedback by hand did not scale. Two thousand essays a week came through, and the product lead, Nikhil, wanted a number he could watch. So the team built a judge: a second model prompt that read the essay and the feedback and returned a score out of ten for "feedback quality", with a sentence of reasoning. The average across a week was 8.4. Nikhil put it on the dashboard.

A new feedback prompt was written that produced longer, more structured feedback with headings and a summary line. On the judge, it scored 8.9 against the old prompt's 8.4. Nikhil wanted to ship it on Monday.

The evaluation engineer, Meera, said the judge had never been checked against a human and that until it had, 8.9 versus 8.4 was one model's opinion of another model's prose. She asked for a week to calibrate it: sixty essays with feedback, labelled independently by two English teachers on the staff, and the judge's verdicts compared against theirs.

Nikhil's objection was cost and time. Two teachers for five hours each was a real expense, the teachers had classes, and a week's delay on a prompt that scored half a point higher looked like process for its own sake. "If the judge says it's better and it reads better, what are we waiting for?"

Meera's answer was that it read longer, and she did not know whether it read better, and neither did he. She also had a specific worry. The judge scored on a 1-to-10 scale, and she had noticed that almost every score was a 7, 8 or 9. A scale where everything lands in the same three numbers carries almost no in-

formation, and a model asked to compare two pieces of feedback tends to prefer the longer, more structured one whatever its content.

The decision was whether to ship the new prompt on the judge's say-so, or to spend the teachers' ten hours first.

Meera also pointed out that the judge and the feedback generator were the same model family, and that models tend to prefer their own family's prose. That was one more reason to doubt the half point, and one more thing a calibration set would settle.

There was a second question underneath, which Meera raised and Nikhil initially waved away: what the score was for. Nobody could say what a move from 8.4 to 8.9 meant in terms of a student's writing. What the teachers actually cared about, when asked, was three specific things. Did the feedback misstate what the student had written? Did it correct a sentence that was already correct? Did the three suggestions address the essay's actual weaknesses rather than generic advice? Each of those is a yes-or-no question about an observable thing, and none of them is a 1-to-10.

What would you have done with the judge, and would you have shipped the new prompt on Monday?

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

The teachers got their ten hours. Sixty essays, labelled blind by both, with a written ruling for each of the nine disagreements. Meera then ran the existing 1-to-10 judge over the same sixty and compared.

Overall agreement, counting a judge score of 7 or above as "acceptable", was 86%. That looked fine and told her nothing. Of the twelve items the teachers had marked as containing a factual misstatement about the student's essay,

the judge had scored nine of them 8 or above. Its recall on the failure the teachers cared most about was 25%.

She rebuilt the judge as three binary rules, one per call, each requiring a quoted span before the verdict: does the feedback state something about the essay that the essay does not contain; does it correct a sentence that was already correct; does each suggestion refer to a specific sentence or paragraph. The quoted spans were checked by code to be genuine substrings of the essay or the feedback, which caught the judge inventing evidence twice in sixty items. Recall on misstatements rose to 83% and the kappa against the teachers was 0.74, which she wrote at the top of the rubric file with the date.

Then she ran the old prompt and the new prompt through the new judge, on the same essays, with each pairwise comparison made in both orders. The new prompt's feedback was 60% longer. Its misstatement rate was the same as the old prompt's. Its suggestions were specific slightly less often, because the extra length was mostly a generic paragraph about structure. The pairwise judge preferred it, and disagreed with itself on 22% of pairs when the order was swapped.

The new prompt did not ship on Monday. Word count and language checks moved into deterministic code. The 1-to-10 score came off the dashboard and was replaced by the three rule pass rates, and the sixty labelled essays became a calibration set that was re-run every month and whenever the judge model changed.

The judge agreed with the teachers on 86% of items. Why did Meera treat that as no evidence that the judge was usable?

Most feedback is acceptable, so a judge that says acceptable nearly always agrees with humans nearly always. The number that mattered was recall on the rare class the teachers cared about, misstatements about the essay, and there the judge caught three of twelve. Per-class recall on a stratified calibration set is the measurement; overall agreement under imbalance is the same trap as accuracy on a rare event.

The new prompt scored 8.9 against 8.4 on the original judge and was preferred in pairwise comparison. What does the 22% self-disagreement on swapped orders say about that preference?

When the judge contradicts itself on 22% of pairs depending only on which answer it saw first, no win margin under about 22 points can be distinguished from position bias. Combined with the new feedback being 60% longer, the preference is better explained by length and structure than by quality, which is why the team measured misstatements and specificity separately from preference.

Why did requiring the judge to quote a span before giving a verdict, and checking that span by code, improve the measurement?

A verdict given first is a commitment the model then justifies; evidence given first conditions the verdict on it. Checking that each quoted span is a real substring is free and deterministic, and it caught the judge fabricating evidence twice in sixty items, a failure that would otherwise have been invisible and counted as a confident judgement.

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 pairwise judge is run on every pair in both orders and contradicts itself on 25% of pairs. What follows?

A. Use the first-order verdicts; the swapped run is only a sanity check on the prompt.

B. Average the two orders' scores so that the position bias cancels out.

C. The judge model is unusable for any purpose and must be replaced.

D. No margin under 25 points means anything; count a win only when both orders agree.

2 Why does the course ask a judge to check six rubric rules in six separate calls rather than all six in one?

A. Because six short calls cost less in total than one long call.

B. Because in one call the verdicts correlate: the model forms an impression and makes all six agree.

C. Because a context window cannot reliably hold six rules together with the policy text and the reply at once.

D. Because separate calls allow the judge to see the reference answer for each rule.

3 A system scores 93% against a set of reference answers. Before concluding anything, the course says to audit fifty disagreements. Why?

A. Because a share of the disagreements, often around a third, turn out to be errors in the references.

B. Because a judge comparing candidate to reference is subject to position bias in the same way as a pairwise judge.

C. Because 93% sits inside the run-to-run spread of most systems.

D. Because references should be written as prose, and structured ones produce false negatives.

4 You have every pairwise comparison between six systems in a file. Why fit a Bradley-Terry model rather than compute Elo ratings?

A. Because Elo cannot represent ties, and ties are informative.

B. Because Bradley-Terry corrects for the length and confidence bias that Elo inherits directly from the raters' preferences.

C. Because Elo updates in match order by a fixed step, so the result depends on order and step size.

D. Because Elo is only defined for more than six competitors.

5 In the judging cascade, why do deterministic checks run before any judge call?

A. They are free, exact and repeatable, and settle 20 to 40% of items outright.

B. Because judge models cannot reliably parse JSON, so structure must be checked first.

C. Because the small local judge needs the deterministic verdict as an input feature before it can decide.

D. Because deterministic checks have higher recall than any judge on quality criteria.

6 An agent task passes on three of five runs. How should it be reported?

A. As passed, with a 60% confidence attached.

B. As not passing; report the mean pass rate and how many tasks passed all five.

C. As passed, since a majority of runs succeeded.

D. As flaky, and removed from the suite entirely so that it cannot distort the average pass rate.

MODULE 6

Evaluation in the working loop

A set that gets run once is a report. This block makes evaluation part of how the work happens: a harness that runs on every change without going red at random, traces complete enough to diagnose from, an error-analysis habit that turns failures into causes, an honest experiment when you want to claim an improvement, and a safe route onto live traffic.

> BY THE END OF THIS MODULE YOU CAN
>
> Run evaluation as part of development — a harness in CI that survives a stochastic model, logging complete enough to diagnose a failure you did not reproduce, a structured error-analysis session that names causes rather than symptoms, and a staged release with a metric that can stop it

50 · 9 min

The harness, and why you should build the small one first

THE ONE THING TO KEEP

A run is reproducible only if the dataset hash, prompt version and model version are recorded with it, and the useful output is a diff against the previous run, not a number.

What a run is

Everything in this course reduces to one operation: take a dataset, run a system over it, score each item, summarise. The harness is the code that does that reproducibly. It is smaller than you think — a first version is about eighty lines — and getting it right early is what makes every later lesson cheap.

A **run** is identified by everything that could change the answer:

```
{"run_id": "2026-09-06T11:04Z-a17c",
 "dataset": "support-v4", "dataset_sha": "9f21c0",
 "system": "reply-agent", "git_sha": "e4b1aa",
 "prompt_version": "reply-v9",
 "model": "claude-haiku-4-5-20260101", "temperature": 0.2,
 "judge_model": "...", "judge_prompt_version": "rubric-v3",
 "seed": 7, "n": 100}
```

If any of those is missing, two runs that disagree cannot be explained, and you will spend an afternoon finding out that somebody edited the prompt. The dataset hash matters as much as the git sha: a changed set with an unchanged name is the most confusing bug in this whole discipline.

Store per item, summarise later

Write one record per item, not just a total:

```
{"run_id": "...", "item_id": "tkt-0041", "output": "...",
 "checks": {"json_valid": true, "language_match": true,
            "r1_no_commitment": false},
 "pass": false, "latency_ms": 1840, "tokens_in": 1204,
 "tokens_out": 210, "cost_usd": 0.0031}
```

Everything else is a query over these. Per-slice pass rates, the bootstrap interval from module one, the flip counts between two runs — all of them need per-item records, and none of them can be recovered from a stored average. JSONL files in a directory are a perfectly good store for the first year; a table in Postgres once you want to query across runs.

Cache, or you will not run it

The difference between an eval you run on every change and one you run monthly is how long it takes. Cache on a hash of everything that determines an output:

```
key = sha256(f"{model}|{temperature}|{prompt_version}|
{item_id}|{input}")
```

Change the prompt and every item re-runs. Add ten items and only those ten cost anything. Re-run to regenerate a report and it is instant and free. A cache turns a twelve-minute, four-dollar run into a nine-second one, and that is the difference between a tool people use and a ritual people avoid.

Keep an explicit `--no-cache` flag, and use it when you want to measure run-to-run variance.

Concurrency, retries and the failures that look like quality

Run items concurrently — twenty at a time is usually safe — and the wall clock drops from twelve minutes to under one. Two disciplines go with it:

- **Retry transient errors** (429, 500, timeouts) with backoff, and **count them separately**. An item that failed because of a rate limit is not a qual-

ity failure, and silently scoring it zero drops your number for reasons that have nothing to do with the system.

- **Never retry a bad answer.** Retrying until something passes is not evaluation.

Where it runs

Three tiers, matching the cost tiers from the previous module:

- **Every pull request:** the deterministic checks on the full dev set. Seconds, free, blocking. If schema validity drops, the build fails.
- **Nightly, or on merge to main:** the judge set. Minutes, a few dollars. Posts a comparison against the previous run — the total, and which items flipped.
- **Monthly or per release:** the test set, plus whatever human review you have committed to. Slow, expensive, decisive.

The first tier is the one that changes behaviour, because it is the only one that catches a mistake before it is merged.

Report the diff, not the number

The output of a run should be a comparison, because a number alone provokes no action:

```
reply-v9 vs reply-v8   (support-v4, n=100, 3 runs)

  pass rate   82.0% -> 86.3%   (+4.3, CI on diff +0.8 to +7.9)
  improved    9 items
  regressed   4 items   <- read these
  R1 gate     0 -> 0 violations
  p95 latency 2.9s -> 4.4s     <- got slower
  cost/item   $0.0031 -> $0.0068
```

Four regressions on a run that improved overall is exactly the information a total hides, and the latency and cost lines stop a quality win from quietly becoming a product loss.

Build the small one first

You will be tempted to adopt a platform. The free ones are good — **promptfoo** (MIT, YAML-driven, up in twenty minutes), **Inspect** (UK AISI, MIT, the most programmable), **DeepEval** (pytest-shaped), **Langfuse** (self-hostable tracing plus evals). The commercial ones — Braintrust, LangSmith — add hosted dashboards and collaboration.

The advice is still to write the eighty lines first. Not because the tools are bad, but because a platform adopted before you know what you are measuring quietly decides what you measure: you will use the metrics it ships with. Write your own loop, learn what your system's failures actually look like, and then adopt a tool to stop maintaining plumbing. Migrating a JSONL set into any of these is an hour.

BEFORE YOU MOVE ON

A nightly eval drops from 87% to 79% overnight with no deployment. What does a well-built harness let you check first?

A. Whether the dataset hash, prompt version and model version string are identical between the two runs.

B. Whether the failures cluster in one slice of the eval set.

C. Whether the provider had an incident during the run window.

D. Whether the judge model has drifted, by re-running the calibration set.

51 · 9 min

An evaluation that runs on every change without going red at random

THE ONE THING TO KEEP

Block merges only on deterministic checks and on a band around a rolling baseline, quarantine flaky items instead of retrying them, and let the slow full run open an issue rather than stop the queue.

Why the obvious design fails

The obvious design is: run the eval set on every pull request, fail the build if the score drops. It survives about a fortnight.

A stochastic model produces a different score on identical code. The build goes red on a change to a README. Somebody adds a retry. Somebody else raises the threshold. Within a month the gate is either ignored or disabled, and the team is back to shipping on feeling — which is worse than never having built it, because everyone believes a gate exists.

The fix is to separate what can be a hard gate from what cannot.

Three tiers

Tier one: deterministic checks, blocking, under sixty seconds. Schema validity, forbidden strings, prompt templates rendering without missing variables, citations resolving, no secrets in outputs, the judge rubric file parsing. These are unit tests. They are reproducible, they never flake, and they catch a genuinely large share of real breakages.

Tier two: a small core set, blocking on a band. 150 to 300 items, one run, temperature 0, pinned model. Do not compare against a fixed threshold; compare against a rolling baseline from the last several runs on the main

branch, and fail only when the drop exceeds a band derived from the measured run-to-run spread:

```
fail if  pass_rate < baseline − max(3 ∗ run_to_run_sd,
minimum_band)
```

Measure that spread once — the A/A test from the statistics block — and write it into the config with a comment saying when it was measured. A gate whose sensitivity is derived from data survives arguments; a gate set at "must be above 85%" does not.

Tier three: the full set, nightly, non-blocking. It runs on a schedule, posts its result, and opens an issue when it regresses. It is where the number you report comes from. It never stops the merge queue, because a two-hour job that blocks a queue gets deleted.

Three tiers, and only two may block

Tier one: deterministic checks	blocking, under sixty seconds: schema, forbidden strings, citations resolve, templates render
Tier two: core set of 150 to 300 items	blocking on a band around a rolling baseline, one run, temperature 0, pinned model
Tier three: the full set, nightly	non-blocking: posts the result, opens an issue on a regression, never stops the merge queue

Tier two fails only when pass_rate < baseline − max(3 × run_to_run_sd, minimum_band). A gate set at 'must be above 85%' is disabled within a month; one derived from the measured spread survives arguments.

Flakiness: quarantine, never retry

There is one rule here worth being strict about. **Never re-run a failing evaluation item until it passes.** That is optional stopping wearing a CI badge, and it converts your gate into a machine for confirming whatever you hoped.

Retries are legitimate for transport failures only — a 500, a timeout, a rate limit — and those should be retried by the client, counted, and reported separately, because a rising retry count is itself a signal.

For items that genuinely flip between runs, quarantine them. An item that changes verdict twice in a week moves to a quarantine list: still run, still reported, no longer blocking. Review the list weekly. A growing quarantine list means either your prompt is unstable or your rubric has an ambiguous case, and both are worth an hour.

Cost and time budgets

Set both explicitly, in the config, as numbers the job enforces:

- Wall clock for the blocking tiers: five minutes. Beyond that, people stop waiting and start merging around it.
- Cost per pull request: a stated ceiling, enforced by the harness, with the run aborting and reporting rather than quietly spending.

Caching by output hash does most of the work here — a pull request that changes documentation should cost nothing to evaluate. Recorded fixtures do the rest: capture real model responses once and replay them for the deterministic tier, so most CI runs need no network access, no API key, and no vendor availability.

Report a diff, not a number

The pull request comment should say which items changed verdict, not what the score was:

```
core set: 268/300 (baseline 271, band -9)  OK
  3 items regressed:  refund-window-hinglish, quote-with-emoji,
empty-input
  1 item improved:    long-thread-truncation
  cost $0.31, cache hit 82%, 2 transport retries
```

Three named items are something a reviewer can open and read. A drop from 90.3% to 89.4% is something a reviewer scrolls past.

Write the override policy down

Sometimes a real regression should ship anyway — a deliberate trade, a fix for something worse. Decide in advance who may override, and require the override to be recorded in the pull request with a reason. An unrecorded override is indistinguishable from a broken gate three months later, when somebody is trying to work out when the quality dropped.

And keep the honest limit in view: a green CI run says the set did not detect a regression. It says nothing about behaviour the set does not contain, which is most behaviour. That gap is what the nightly run, the production monitoring and the incident-to-item pipeline are for.

BEFORE YOU MOVE ON

An evaluation job in CI is failing intermittently on the same three items. Which response is defensible?

A. Add a retry loop so each item is attempted up to three times and passes if any attempt succeeds.

B. Raise the pass threshold band until the intermittent failures no longer trip it.

C. Move those items to a quarantine list that still runs and reports but does not block, and review why they flip.

D. Remove the three items from the set, since unstable items cannot contribute to a reliable score.

52 · 9 min

Recording enough to diagnose a failure you did not see happen

THE ONE THING TO KEEP

Log the whole trace — the exact prompt sent, the retrieved chunks, the tool calls, the versions, the timings and the user's next action — because every production failure is diagnosed from what was recorded and never from what you can reproduce.

The complaint arrives without its context

A user says the assistant gave them the wrong refund window on Tuesday. You cannot reproduce it: the corpus has changed, the prompt has changed, and you do not know what they typed. Whether that complaint becomes a fix or a shrug was decided weeks earlier, by what the system records.

The minimum useful record

Per request, one trace, containing:

- **Identity and time** — request id, timestamp, session id, a hashed user id, and the surface it came from.
- **The exact prompt sent to the model.** Not the template. The rendered string, or the message array. This is the single most valuable field and the one most often missing.
- **Versions** — model id including the dated variant, prompt template version, rubric version, retriever and index version, application build.
- **Parameters** — temperature, top-p, max tokens, tool schema version, seed if you set one.
- **Retrieval** — chunk ids, their scores, and either the chunk text or a content hash pointing at an immutable store.

- **Tool calls** — the name, the arguments, the result, the duration, the errors.
- **The output**, plus token counts in and out, cost, and the finish reason. A truncation is invisible without the finish reason and explains a startling share of "the answer just stops" reports.
- **Timings per span** — retrieval, generation, judging, post-processing.
- **What the user did next** — regenerated, edited, copied, escalated, abandoned. This is the free quality signal, and it only exists if you attach it to the trace id.

Spans turn "it was slow" into a fix

Structure the record as a trace with nested spans rather than one flat row. A user complaint of slowness becomes "retrieval 2.1 s of 2.6 s total, of which 1.8 s was the embedding call", which is a specific thing to fix in an afternoon. Without spans it is a week of guessing.

OpenTelemetry has conventions for this and most tooling speaks it. Self-hosted open-source options exist and are worth the day it takes to run one. And if that is more than you want to operate: a Postgres table with a `jsonb` column, a few indexed fields and a retention job is genuinely sufficient for a small product. The failure is not choosing the wrong tool; it is having nothing.

Sample the trace, not the fields

Full traces are large — context, chunks, outputs — and storage can exceed inference cost if you keep everything for a year. Sample deliberately:

- 100% of errors, timeouts and guardrail hits.
- 100% of sessions where the user escalated, edited heavily, or reported a problem.
- A fixed percentage of everything else, chosen so the sampled stream supports your monitoring arithmetic.

Make the sampling decision once per trace and propagate it, so you never end up with a request whose retrieval span was kept and whose generation span

was dropped.

Logs of user text are your most sensitive store

Treat them accordingly, because they are exactly what a breach report would be about.

- Redact identifiers at ingestion, and accept that automatic redaction is imperfect — it will miss the sentence where somebody names their employer.
- Set a retention period and enforce it with a job, not a policy document.
- Restrict access and log the access.
- Keep the legal basis straight: what your privacy notice promised about using conversations to improve the product is what governs whether this data may become an evaluation set at all.

Where policy is strict, log hashes and structure rather than content: chunk ids instead of chunk text, a redacted skeleton instead of the message, lengths and categories instead of the string. It is a real reduction in diagnostic power, and it is better than a blanket prohibition that leads people to keep copies elsewhere.

The button that pays for the whole system

Every trace is a potential evaluation item. Build the one-click path: from a trace in the viewer, create an eval item with the input, the retrieved context, the produced output, and a field for what the answer should have been.

That single affordance is what makes the incident-to-item habit in the last block practical rather than aspirational. Without it, adding a real failure to the set means twenty minutes of copying, and it does not happen.

BEFORE YOU MOVE ON

A team logs the prompt template id and the variables, but not the rendered prompt. What goes wrong first?

A. Storage costs rise, because templates and variables together are larger than the rendered string.
B. A failure cannot be reproduced once the template has changed, because the recorded id now renders differently or no longer exists.
C. Token counts become impossible to compute, since they depend on the rendered text.
D. The judge cannot be run retrospectively on the logged output, since it needs the template.

53 · 8 min

Regression testing a prompt

THE ONE THING TO KEEP

Track which items flipped, not just the total; an unchanged score can hide a rewritten system.

A prompt is code with no type system

Every edit is a refactor with unknown blast radius. You add one line to fix how it formats Indian dates and it starts writing Mexican peso amounts with a full stop where the comma should be. Nothing errors. Nothing warns you. The change reaches production and looks fine for three weeks.

So treat prompt edits the way you treat edits to a function with a thousand callers: run the tests.

The setup, concretely

Put the eval set in the repository. Add a command — `npm run eval`, `pytest tests/eval` — that runs the frozen set against the current prompt and writes results per item. Wire it into CI on any pull request that touches a prompt file, a rubric, or a model id.

Keep a fast lane and a slow lane. Ten items that run in twenty seconds for local iteration, the full hundred-plus in CI. Cache results keyed on a hash of `(prompt, input, model, temperature)` so re-running after an unrelated change costs nothing.

Pin everything. An explicit dated model id, never an alias like `-latest`. Temperature 0 for evals, while being honest that temperature 0 is not actually deterministic — for items that flip between runs, run them three times and record the flakiness rather than pretending it away.

The report that matters is not the score

This is the part teams get wrong. Your run prints 84%, the previous run printed 84%, and everyone moves on. But an aggregate can hold perfectly still while a third of the individual verdicts change places.

So print the flips:

```
eval: address-eval-100    84% -> 84%   (no change)

  newly passing (7):  addr-003 addr-018 addr-044 addr-051
                      addr-067 addr-072 addr-090
  newly failing (7):  addr-009 addr-021 addr-033 addr-058
                      addr-061 addr-079 addr-095

  by tag:
    landmark-only   61% -> 44%   (-17)
    street-address  92% -> 99%   (+7)
```

Now you can see it. You did not make a neutral change; you traded away the landmark cases to gain the easy ones. That trade might be correct — but it

should be a decision, not an accident, and the aggregate would have hidden it completely.

84% to 84%, and fourteen verdicts moved

	Passes now	Fails now
Passed before	**77** unchanged	**7** newly failing
Failed before	**7** newly passing	**9** unchanged

Seven items in, seven out, and the total holds perfectly still. By tag, landmark-only fell from 61% to 44% while street-address rose from 92% to 99%. That is a trade, and the aggregate hid it.

Golden outputs in the diff

The highest-value habit in this whole lesson: commit the actual generated outputs for a small set of items, so a prompt change shows up as a readable text diff in the pull request.

```
- Your refund will be processed within 5–7 business days.
+ ## Refund status
+ Your refund will be processed within 5–7 business days.
+ Let me know if there's anything else I can help with.
```

No metric caught that. A reviewer catches it in one second, and it is the kind of change — headings appearing, sign-offs appearing, personality drifting — that annoys users long before it moves a score.

Thresholds, used lightly

Fail the build on a drop of more than a few points, sure. But a hard gate on a hundred-item set will produce false alarms from normal variance, and a team that gets three false alarms stops reading the output entirely. The gate is a prompt to look, not a verdict. The flip list and the golden diff are what you actually read.

What this buys you

Speed, mostly. With this in place you can make a risky prompt change on a Tuesday afternoon and know within four minutes whether it broke the Portuguese cases. Without it, every change is a small act of faith, and after enough of them nobody on the team is willing to touch the prompt at all.

BEFORE YOU MOVE ON

After a prompt edit, the eval run reports 84% before and 84% after on the same frozen set. What should the team conclude?

A. Nothing yet — the same total is produced both by "nothing changed" and by a batch of items swapping places, and only the per-item diff distinguishes them.

B. The edit is behaviour-neutral on their test set, so it is safe to ship.

C. The edit had no effect and should be reverted to keep the prompt short.

D. The set is too small to detect the change and should be expanded to 300 items before deciding.

---

54 · 9 min

Reading fifty failures

THE ONE THING TO KEEP

Read fifty failures, write the cause in your own words before categorising, and label each cause as data, retrieval, prompt, model, label or product — most of them will not be prompt problems.

The highest-value hour in applied AI

Sit down with fifty failures. Read each one. Write one line saying why it failed. Then group the lines.

That is the whole technique, it takes about ninety minutes, and it will change what you work on next more than any other activity in this course. Teams that do it routinely stop guessing; teams that do not spend months rewriting prompts against problems that were never prompt problems.

The reason it works is the distribution. Failures are never spread evenly across twenty causes. Three causes are usually 60 to 80 per cent of them, and you cannot see which three from a summary metric.

Do it in this order

1. Take failures, not a random sample. You are diagnosing, not estimating a rate. Fifty is plenty; twenty is enough to see the big clusters.

2. Write the cause in your own words, before naming a category. "Only the first page of the receipt was in the input" — not "extraction error". Free text first. Categories imposed too early hide the thing you did not expect.

3. Cluster afterwards. Sort the fifty lines, merge the ones that say the same thing, count.

4. Assign each cluster an owner and a *kind*. This is the step that pays. Kinds:

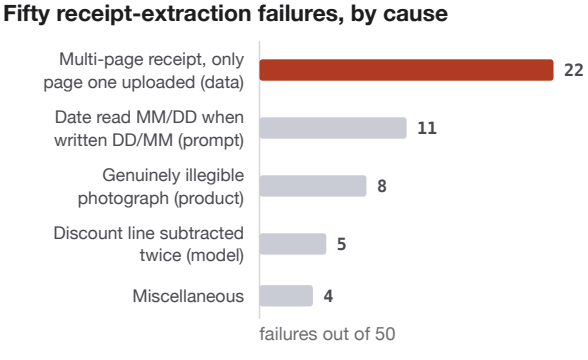
- **Data** — the input was wrong, missing, or not what the system was given.
- **Retrieval** — the information was not supplied.
- **Prompt** — the instruction was ambiguous or absent.
- **Model** — the instruction was clear and it did not comply.
- **Label** — the expected answer was wrong. It happens more than teams expect.
- **Product** — the request was reasonable and the system is not designed to serve it.

What it looks like

A receipt-extraction system, 50 failures:

- **22 — multi-page receipts, only page one uploaded.** *Data*. The mobile client sent one image. No prompt change would have helped.
- **11 — dates read as MM/DD when written DD/MM.** *Prompt*. One sentence naming the locale convention, plus a deterministic check that the parsed date is not in the future.
- **8 — genuinely illegible photographs.** *Product*. The answer is to detect blur and ask for a retake, not to extract harder.
- **5 — totals wrong because a discount line was subtracted twice.** *Model*. Real, and the smallest cluster.
- **4 — miscellaneous.**

The team had been comparing three prompt variants for a fortnight. Two of the three top causes were not in the prompt at all, and the biggest one was a client bug. That is the typical shape of this exercise, not a dramatic example chosen to make a point.



The team had spent a fortnight comparing three prompt variants. The largest cause was a client bug, and only five of the fifty failures were the model's.

Turn the clusters into a taxonomy

Write the cause names down and reuse them. Tag every future failure with one. Within two months you have a chart of failure causes over time, and it answers the question no pass rate can: *what is going wrong now, and is it the same thing as last month?*

A cluster that keeps reappearing after a fix was shipped means the fix did not work, which is exactly the fact that a stable overall score conceals.

Cheap tricks that speed it up

- **Sort failures by input length, language, or slice tag** before reading. Clusters often fall out of the ordering for free.
- **Cluster automatically for a first pass.** Embed the failure outputs with a free local model, run k-means with k around 8, read three items per cluster. This is a way to *find* candidate groups quickly, not a substitute for reading. The names you get from reading are better than the ones you get from a centroid.
- **Diff against a run that passed.** For items that used to pass and now fail, put the two outputs side by side. The cause is usually visible in seconds.
- **Keep the fifty.** They become eval items with an `origin` field, which is the subject of a later lesson.

The rule

No prompt change without error analysis first. It sounds bureaucratic and it is the opposite: it is what stops you spending a fortnight on the 10 per cent cause while the 44 per cent cause sits in a client bug nobody has looked at.

And it is the part of this work that does not automate. A model can cluster your failures; it cannot notice that the receipts are all from one chain whose printer cuts off the total. That noticing is the job.

BEFORE YOU MOVE ON

A team's error analysis of 50 failures finds 21 caused by truncated inputs, 12 by an ambiguous instruction, 9 by genuine model errors and 8 miscellaneous. They have one week. What should they do?

A. Work on the 9 model errors, since those are the only failures the model itself is responsible for.
B. Fix the truncation in the input pipeline first, then the ambiguous instruction, and leave the model errors for later.
C. Rewrite the prompt to handle all four clusters at once, since a single prompt revision is cheaper than four separate fixes.
D. Collect another 200 failures first, since 50 is too small a sample to allocate a week of work against.

55 · 8 min

A/B testing a feature honestly

THE ONE THING TO KEEP

Fix the metric, the effect you would act on, and the stop date before you start; otherwise the result is a story.

What an A/B tells you that your eval set cannot

Your eval set answers "is this better on the cases I chose, by the criteria I wrote". An A/B answers "do people behave differently". Both matter and neither substitutes. A rewrite can improve every criterion on your set and reduce the number of people who finish the flow, because it now sounds like a lawyer.

An honest A/B is mostly about decisions you make before it starts.

Write it down first, in one paragraph

```
Experiment: reply-rewrite-v4
Primary metric: share of conversations resolved without
                a human handover
Minimum effect we would act on: 3 percentage points
Guardrails: p95 latency, complaint rate, moderation reports
Allocation: 50/50, new sessions only
Runs: 2026-09-08 to 2026-09-22. No decision before the 22nd.
```

Five lines. They are the difference between an experiment and a story you tell afterwards.

Peeking is the common way this goes wrong

You plan two weeks. On day four the metric is ahead with p = 0.03. You ship.

That p-value assumed you would look once, at the end. Looking every day and stopping the first time you cross the line means you get many chances to cross it by luck, and the real false-positive rate for a daily-peeked two-week test lands somewhere around 20 to 30% rather than 5%. Most "our A/B said it worked but it did nothing in production" stories are this.

Two honest fixes. Fix the horizon and genuinely do not decide early. Or use a method built for continuous monitoring — sequential tests, always-valid confidence intervals — which trade a little sensitivity for the right to look whenever you want. Pick one before you start.

Guardrails, because your primary metric is a proxy

Resolution rate goes up. Also: median latency went from 900ms to 2.4s, complaints mentioning "robotic" doubled, and unsubscribes ticked up. You would never have seen those, because you were watching one number.

Pick two or three guardrails that represent the ways a win could be hollow, and check them even when the headline is good. Especially when the headline is good.

Segments: declare them before, not after

The result is flat. Someone slices the data and finds it worked for Android users in Brazil. That finding is almost certainly noise — slice enough ways and something is always significant. If you had a reason in advance to expect a difference for a segment, declare it in advance and analyse it. Anything found afterwards is a hypothesis for the next experiment, never a conclusion from this one.

When you do not have the traffic

Many products cannot run a meaningful A/B. Four hundred sessions a week will not detect a three-point change this quarter or next. Say so out loud rather than running an underpowered test and treating its noise as a signal.

What works at small scale:

- **Interleaving**, where both variants serve the same user and you compare which one gets used. Far more sensitive per user, though it only applies to ranked or multi-option outputs.
- **A long, boring baseline.** Six weeks of before, six weeks of after, with your eyes open about everything else that changed (a holiday, a marketing push, a competitor's outage).
- **Ten conversations with real users.** Not statistics, and enormously more informative than a test with no power. Ten people telling you the new replies feel like a form letter is a finding.

The least honest option is the underpowered test, because it produces a number, and a number outranks a hunch in most meetings even when the hunch is better evidence.

BEFORE YOU MOVE ON

A team plans a two-week A/B and checks the dashboard daily. On day four the new variant is ahead at p = 0.03, so they stop and ship. What is the core error?

A. Stopping the moment a threshold is crossed changes what the threshold means, so their real chance of a false positive is much higher than 3%.
B. Nothing is wrong — significance is significance, and stopping early spares users the worse variant.
C. Four days is too short because weekday and weekend users behave differently.
D. p = 0.03 is too weak a threshold for a shipping decision; they should have required p < 0.01.

56 · 8 min

Getting evidence from live traffic without exposing users

THE ONE THING TO KEEP

Run the new system on real inputs with its side effects disabled to compare distributions before anyone sees it, then ramp real traffic in stages with preagreed guardrails and a rollback you have rehearsed.

Three stages between the eval set and everyone

An offline set says the change is probably good. Live traffic says whether it is. The gap between them is crossed in three steps, each answering a different question.

Shadow — the new system processes real inputs and its outputs are discarded. Answers: does it behave sanely on the real distribution.

Canary — a small share of real users get the new system. Answers: does anything break at volume, and do the guardrails hold.

Staged ramp — the share grows on a schedule with soak time. Answers: does it survive a full day, a Monday, a month-end.

Shadow mode, and the incident everybody has once

Shadow runs are cheap evidence and they have one sharp edge: **side effects must be disabled**. The new agent must not send the email, write the record, call the payment API or post the message. Every team that skips this learns it in the same way, and the second copy of a customer email is the polite version of the lesson.

Enforce it structurally, not by convention: a shadow request carries a flag that the tool layer checks, and effectful tools return a simulated result. Then verify it, by running the shadow path against a staging tool endpoint that records calls and asserts none arrived.

What shadow gives you, with no labels at all:

- **Distribution comparisons on real inputs** — output length, refusal rate, groundedness rate, schema validity, tool-call frequency, cost, latency percentiles. A shift in any of these on real traffic is a real finding.
- **A paired comparison on thousands of items**, since old and new ran on identical inputs. Every label-free check from earlier in the course applies directly.
- **Error discovery** — the inputs that crash the new path, which your set never contained.

It cannot tell you whether users prefer the result, and it doubles inference cost while it runs. Run it for a day or two on a sample, not indefinitely on everything.

Canary, sized by the guardrail

The share is chosen by arithmetic, not by feel: how many events do you need before the guardrail metric could plausibly trigger. If your guardrail is a safety-violation rate that occurs in one request per thousand, a 1% canary on 50,000 daily requests gives you 500 requests a day and roughly one expected

event — far too slow. Either raise the share or pick a guardrail with a higher base rate.

Two practical rules. **Keep assignment sticky** per user, or somebody gets two different behaviours in one session and reports a bug that is really your experiment. And **check whether the canary population is representative** — a canary routed to one region, one platform or one data centre is a biased sample, and the mix-shift arithmetic from the statistics block applies exactly.

Guardrails, thresholds and a rehearsed rollback

Write these before launch, with numbers:

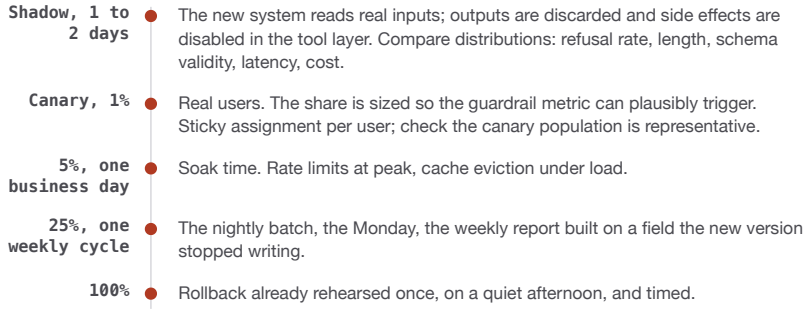
- The metrics that can stop the rollout — error rate, safety violations, refusal rate, p95 latency, cost per request, a business guardrail such as escalation rate.
- The threshold and the window for each, sized so that normal fluctuation does not trigger them daily.
- Who is paged, and what they are authorised to do without a meeting.

Then rehearse the rollback. A rollback that has never been executed is a hypothesis, and the moment you discover the config takes ten minutes to propagate should not be during an incident. Roll back once on purpose, in the middle of a quiet afternoon, and time it.

The ramp, and why soak time matters

1%, 5%, 25%, 100%, with a minimum soak at each step — usually a full business day, and at least one weekly cycle before the final step. Some failures only appear with time or volume: rate limits at peak, cache eviction under load, a nightly batch that now takes longer than its window, a weekly report built on a field the new version stopped writing.

From the eval set to everyone



This machinery detects fast, visible harms. Slow ones, such as a corpus filling with the system's own output or users quietly giving up, need the monitoring in the final module.

The limit worth stating: this machinery detects fast, visible harms. It does not detect slow ones — users gradually trusting an answer that is subtly wrong, a corpus filling with the system's own output, a group of users quietly giving up. Those are the subject of the final block, and no ramp schedule substitutes for them.

BEFORE YOU MOVE ON

What is the specific risk that makes shadow mode dangerous if implemented carelessly?

A. It doubles inference cost, which can exhaust a budget before the comparison is complete.

B. Shadow traffic can trigger rate limits that degrade the live path serving real users.

C. The shadow system executes real side effects — sending messages, writing records, calling payment APIs — because only its returned output was discarded.

D. Shadow outputs are never seen by users, so quality problems can persist unnoticed for weeks.

57 · 9 min

The signals users give you without being asked

THE ONE THING TO KEEP

Regeneration rate, edit distance and escalation are free quality proxies you probably already log, but they only become metrics once you have checked them against human labels on a sample.

Your offline set cannot see everything

A fixed set of a hundred items measures the inputs you collected, against the criteria you wrote, at the moment you wrote them. It cannot see a question nobody has asked yet, a use you did not anticipate, or the slow shift in what people bring to the product. Production can.

The trouble is that production has no labels. So you use proxies: behaviours that correlate with quality and cost nothing to collect.

Implicit signals, best first

Regeneration or retry rate. The fraction of responses where the user immediately asks again, rephrases, or hits regenerate. This is the single best free signal most products already have in their logs and do not look at. It needs no survey and no annotation, it is unambiguous, and it moves quickly when quality moves.

Edit distance between suggestion and what shipped. For anything that drafts text a human sends, compare the draft to the sent version. Accepted verbatim, lightly edited, heavily edited, discarded. This is a graded quality signal collected automatically, and it is the metric that convinces a sceptical team, because it is measured in their own work.

Copy rate, and what gets copied. If people copy the code block and never the explanation, the explanation is not earning its tokens.

Abandonment. Session ends without the task completing. Noisy — people are interrupted — but a sharp change in it is real.

Escalation to a human. For anything with a support path, the fraction of conversations that end with a handover is close to a direct quality measure, and the business already tracks it.

Time to resolution. The lagging metric that usually is the actual point.

Follow-up sentiment. The user's *next* message after a response. "no, I meant", "that's not right", "thanks" — a small classifier over next messages gives a usable dissatisfaction rate, and it is far more representative than a thumbs-down widget.

Explicit feedback, and its limits

Thumbs up and down get roughly one to two per cent response rates, and the responders skew — people click when annoyed or delighted, rarely when satisfied. Treat the *rate* as a trend, never as a level, and read the free-text comments as a source of eval items rather than as a metric.

One thing that improves them a lot: ask at the moment of consequence rather than after every message. A single "did this solve it?" at the end of a resolved conversation gets a far better response rate and a much less skewed sample than a widget on every turn.

Guardrails

Every online experiment needs metrics that are not allowed to get worse, watched independently of the metric you are trying to move:

- p95 latency
- cost per session
- escalation rate
- error and timeout rate

- refusal rate

Refusal rate belongs on that list for the reason module three gave: it is where a safety improvement silently becomes a product regression, and it is the one nobody thinks to watch.

Connect the online signal to the offline set

This is the step that turns proxies into a working system. Take a week of production traffic. For a sample of 100 sessions, compute the online signal (regenerated or not) and also have a human label the quality. Then check whether the two agree.

If regeneration correlates with your quality label, you have earned the right to treat regeneration rate as a quality dashboard, and you can watch it hourly. If it does not — sometimes people regenerate because they want a different *style*, not because the answer was wrong — you have learned that too, cheaply, and before you built a dashboard on it.

The same sample gives you the other direction: sessions where the offline criteria would have passed but the user clearly did not get what they needed. Those become new eval items and new rubric rules. This is the main channel by which an eval set stays relevant to the product rather than to its own history.

Log for it, and log carefully

None of these signals exist unless someone logs them. The minimum, per interaction: a session id, a turn index, the model and prompt version, latency, tokens, cost, whether it was regenerated, and whether the session escalated.

And log with the privacy rules in mind from the first day — redact or hash identifiers, set a retention window, and make sure the people whose messages you are storing have been told. A logging scheme retrofitted for privacy after launch is a much more expensive project than one designed that way, and it is not only a legal question: a team that is nervous about its own logs stops looking at them, which costs you every signal in this lesson.

BEFORE YOU MOVE ON

A team wants an ongoing quality signal for a drafting assistant without paying for continuous human review. Which is the strongest starting point?

A. A thumbs-up/thumbs-down widget on every response, with the daily average as the dashboard metric.
B. The edit distance between each generated draft and the version the user actually sent, validated against human labels on a sample of 100.
C. Daily token volume and session counts, as a measure of how much value the feature is delivering.
D. An LLM judge scoring every production response against the rubric, with the mean score charted hourly.

58 · 9 min

Sampling live traffic, and keeping reviewers honest

THE ONE THING TO KEEP

Review a random sample as well as a flagged one, because the flagged sample only ever teaches you about failures you can already detect.

Automation has a floor

Deterministic checks catch the mechanical failures. A calibrated judge catches the criteria you have already written down. Neither catches the failure nobody has thought of yet, and that is the one that ends up on social media.

So some human reading of live traffic is permanent. The engineering question is how to get the most out of a small, fixed amount of it — realistically one person for an hour a day, not a review team.

Two samples, never one

A random sample. Twenty items a day, drawn uniformly from all traffic, no filtering. This is the only thing that gives you an unbiased error rate and the only thing that finds unknown failure modes. It is also boring, because most of it is fine, and it is therefore the first thing cut. Protect it.

A targeted sample. Twenty more, drawn from where the problems are: items the judge flagged, items with low retrieval scores, items that were regenerated, sessions that escalated, anything a user complained about. This is where you find the most bugs per minute.

Review only the targeted sample and you learn about the failures you can already detect, which is a closed loop that congratulates itself. The random sample is what breaks it open. A useful diagnostic: count how many *new* failure kinds each sample produced this month. When the random sample stops producing new kinds, your automated checks have genuinely caught up, and that is worth knowing.

Forty items a day is a real programme

Do the arithmetic before designing anything elaborate. Two minutes per item, forty items, is eighty minutes a day. Over a month that is roughly 900 reviewed items, which at typical volumes is a one to two per cent sample and enough to estimate an error rate to within about three points.

That is a modest, sustainable commitment that most teams can actually keep, and it beats an ambitious review process that is abandoned in week three.

Keep the reviewers calibrated too

Reviewers drift. They get faster, they get more forgiving, they develop private conventions. Three cheap controls:

- **Overlap 10%.** One item in ten goes to two reviewers. Track agreement over time. A falling kappa is drift, and it shows up weeks before anyone would have noticed by feel.

- **Salt the queue.** Every so often, insert an item that was labelled months ago and whose answer is known. If today's verdict differs from the archived one, the standard has moved, not the traffic.
- **Run a short calibration meeting.** Half an hour a fortnight, five disagreements, and a written ruling added to the guide for each. This is the only mechanism that keeps the guide alive.

Design the queue so it is quick to be right

The interface decides the throughput more than the reviewer's speed does. The rules that matter:

- **One item, one screen.** The input, the output, the retrieved context, and the buttons. No scrolling to reach the verdict.
- **Ask the rubric's questions, not "is this good".** Four checkboxes matching your rules, not a five-star widget. Same reason as in module three: a fused score cannot be acted on.
- **A free-text box for "something else was wrong".** This field is where new failure modes come from. Read it weekly, and treat a recurring phrase in it as a candidate rubric rule.
- **Show the item's provenance** — random or targeted, and why it was flagged — but *after* the verdict, not before, or you have primed the reviewer.

Label Studio and **Argilla** are both free and open source and do all of this out of the box, including blind double-review and agreement reports. A spreadsheet with a form front-end works at forty items a day and costs nothing at all.

The output is not a score

The point of the review programme is not the weekly quality percentage. It is three things:

1. **New eval items**, with expected outputs, added to the set.
2. **New rubric rules**, from failures your criteria did not cover.

3. **A calibration signal for the judge**: reviewed items are exactly the labelled data your judge should be re-measured against, at no extra cost.

That third point closes the loop cleanly. Live review generates labels; labels calibrate the judge; the judge scales to all traffic; its flags target tomorrow's review. Each layer feeds the next, and the human hour a day is what keeps the whole thing anchored to reality rather than to its own past.

One caution to end on. Reviewing your own product's failures every day is dispiriting if it is framed as quality control on people. Frame it as the team's route to the next fix, rotate it, and share what each week's review changed. A review programme that nobody wants to do lasts about a month.

BEFORE YOU MOVE ON

A team reviews 50 production items a week, all drawn from cases their judge flagged as low quality. After three months their error rate looks stable and no new failure types have appeared. What is the most likely reason?

A. The system has genuinely stabilised, and the review budget could now be reduced.

B. The sample only contains failures the judge already recognises, so it cannot surface anything the judge misses.

C. Fifty items a week is too small a sample to detect new failure types at typical rates.

D. Reviewer drift has made the reviewers more lenient over three months, hiding a real decline.

CASE STUDY · MODULE 6

The gate that went red on a README

A Hyderabad startup running a WhatsApp support assistant for a broadband provider, with a CI gate nobody trusted and an A/B test the CEO wanted to call on day four

Netra Support runs the WhatsApp assistant for a regional broadband company with about 300,000 subscribers. The assistant answers billing questions, books technician visits and explains plan changes, and it hands over to a human agent when it cannot. The team of six had done the early work properly: a 200-item eval set from real chats, a rubric, a judge, and a CI job that ran the set on every pull request and failed the build if the pass rate fell below 85%.

Within a month the gate was a joke. The model was stochastic, so the score moved two or three points between identical runs. A pull request that changed only the README went red. A developer added an automatic retry. Another raised the threshold to 83. By the time the founder, Sameer, noticed, the job had been marked "allow failure" and nobody had looked at its output for three weeks.

Then a customer was told the wrong window for a refund, twice, and posted the screenshots.

Two things were in front of the team at the same time. The first was what to do about the gate. The engineering lead, Farah, wanted a week to restructure it into tiers: deterministic checks that blocked, a small core set that blocked only when the drop exceeded a band derived from the measured run-to-run spread, and the full set nightly and non-blocking. Items that flipped between runs would be quarantined rather than retried. Sameer's view was that a week of the lead engineer's time on test infrastructure, in the fortnight after a public incident, was a week not spent on the incident.

The second was the A/B test. A rewritten reply prompt had been in a 50/50 experiment for four days of a planned fourteen, with resolution-without-handover as the primary metric and a pre-registered minimum effect of three points. On day four it was six points ahead with a p-value of 0.03. Sameer

wanted to ship it that afternoon, partly because it would be something positive to say publicly. Farah said the p-value assumed one look at the end and that a daily-checked fortnight ran a false-positive rate nearer a quarter than a twentieth. Sameer said he did not care about the statistics if the customers were happier.

Farah did one more thing before the argument reached a decision. She and a colleague took fifty failed items from the last full run and read them, writing one line each about why, before assigning any category. Twenty-one of the fifty were the assistant quoting a refund window that had been correct until the company changed its policy in June; the knowledge base article had been updated, the retrieval index had not. Eleven were the eval set's expected answer being wrong for the same reason. Nine were the assistant answering in English when the customer had written in Telugu. Six were genuine model failures on the prompt. Three were miscellaneous.

So the incident had not been a prompt problem, and the A/B was comparing two prompts on top of a stale index.

What would you ship, what would you fix first, and what would you do with the day-four result?

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 fixed the index. The June policy change had been indexed into a separate namespace nobody was retrieving from; the fix was a configuration line and a re-index, done that evening. The eleven stale expected answers in the eval set were corrected, with a note in the set's changelog and a policy version attached to every refund item, so that the next policy change would fail the run loudly instead of silently penalising the right answer.

The A/B ran to its fourteenth day. With the index fixed underneath both arms, the rewritten prompt's advantage settled at 1.8 points, below the three-point threshold they had written down before starting. It did not ship. Sameer, to his credit, said in the retrospective that the day-four number would have shipped a prompt that did nothing and let the index stay broken for another fortnight.

The gate got its week, but in the second fortnight. Deterministic checks became the only hard block: schema, the language of the reply matching the language of the customer, no refund window mentioned unless it appeared in the retrieved policy text, citations resolving. The core set of 150 items blocked only when the pass rate fell more than three standard deviations of the measured run-to-run spread below a rolling baseline. Nine items that had flipped in the last month went to quarantine, and reading them found two rubric ambiguities. The nightly full run posted a diff of which items changed verdict, not a score, and the Telugu failures became a named slice with its own floor.

The team had retries and a raised threshold on the CI gate before anyone decided to. Why does a fixed threshold on a stochastic eval fail in that specific way?

A stochastic model gives different scores on identical code, so a fixed line is crossed by noise, and every red build on an unrelated change teaches the team to route around the gate. Retrying a failing item until it passes is optional stopping inside CI. The fix is to block only on deterministic checks and on a band sized from the measured run-to-run spread, and to quarantine items that genuinely flip rather than retry them.

On day four the experiment was six points ahead at p = 0.03. Why was Farah right that this did not justify shipping, and what did the final result show?

The p-value was computed as if the team would look once, at the fixed horizon. Looking daily and stopping at the first crossing raises the false-positive rate several-fold and biases the reported effect upwards, because you stop at a high point. Run to the horizon, and with the stale index fixed under both arms, the advantage was 1.8 points, below the pre-registered three-point threshold.

Error analysis showed only six of fifty failures were the model's. How did reading the failures change what the team worked on, compared with what the CI score alone would have told them?

The score said the assistant was worse; it could not say why. Reading fifty failures and writing the cause before categorising showed 21 retrieval failures from a stale index, 11 wrong expected answers in the set, and 9 language mismatches, none of which a prompt change addresses. The fix was a configuration line and a re-index, done in an evening, and a deterministic language check that now blocks merges.

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 Two runs of the same harness disagree by six points and nobody can say why. Which missing record does the course single out as the most confusing cause?

A. The random seed, which the harness lesson says is irrelevant once temperature is zero.
B. The dataset hash: a changed set with an unchanged name.
C. The number of concurrent workers used for the run.
D. The wall-clock time at which the run started.

2 An item in the CI core set flips verdict twice in one week. What does the course prescribe?

A. Retry it until it passes, since transient failures should not block a merge.
B. Delete it, because an item that flips carries no information about the system and only adds noise.
C. Quarantine it: still run and reported, no longer blocking; review the list weekly.
D. Widen the threshold band until the flip no longer causes a red build.

3 Which field does the tracing lesson call the single most valuable in a trace, and the one most often missing?

A. The name of the prompt template that was used for the request.
B. The user's thumbs-up or thumbs-down rating of the response.
C. The model's own stated confidence in its answer.
D. The exact rendered prompt sent to the model.

4 A prompt edit leaves the eval score at 84% before and 84% after. What can hide behind an unchanged total?

A. Nothing of substance; an unchanged score is evidence of an unchanged system.

B. Equal numbers newly passing and newly failing, and a trade between slices.

C. Only run-to-run noise, which by chance came out to the same figure.

D. A cache serving the previous run's outputs, so the new prompt never actually ran.

5 In error analysis, why write the cause of each failure in your own words before assigning it a category?

A. Because categories imposed too early hide the thing you did not expect.

B. Because free text is easier for an embedding model to cluster automatically.

C. Because the taxonomy does not exist until the first fifty have been categorised.

D. Because categories should be assigned by the judge model, not by a person.

6 In shadow mode, what must be enforced structurally in the tool layer rather than by convention?

A. That the new system's outputs are written to the same log store as the old system's, with the same retention.

B. That assignment to the shadow path is sticky per user across the session.

C. That side-effecting tools return simulated results for requests carrying the shadow flag.

D. That the shadow run lasts at least one full weekly cycle before a canary begins.

MODULE 7

Evidence other people can act on

At some point the number leaves your team. A buyer, an auditor, a regulator, a journalist or a user asks what you know and how you know it. This block is about evaluation as a record rather than a result: what goes in a system card, what to ask a vendor before you build on them, what the emerging standards ask for, how to measure harm rather than error, and where measurement itself runs out.

> BY THE END OF THIS MODULE YOU CAN
>
> Assemble an evaluation record a stranger can audit — documented set, per-group results, stated limitations, reproducible run — interrogate a supplier's claims with a set of your own, and say clearly where a quality judgement is contested rather than merely unmeasured

---

59 · 9 min

The document that travels with the system

> THE ONE THING TO KEEP
>
> A system card is only worth writing if every claim in it carries the measurement behind it and the out-of-scope section is specific enough to stop somebody using the thing wrongly.

What it is for

A system card is a short document that goes wherever the system goes: what it does, what it was measured on, where it fails, and who to contact. The idea

comes from model cards and from datasheets for datasets, both proposed around 2018 and 2019, and it has become ordinary practice for published models.

Its purpose is narrow and useful. Somebody who did not build the system has to decide whether to rely on it. The card is what they read.

The sections that earn their place

Intended use. The tasks it was built and measured for, in plain sentences. "Answering questions about our published help articles, in English and Hindi, for logged-in customers."

Out of scope. The most valuable section and the one most often skipped. Be specific enough to stop somebody: "not for medical, legal or financial advice; not for eligibility decisions; not for languages other than the two listed; not for documents the retrieval corpus does not contain."

The data. Where the evaluation items came from, when they were collected, how they were sampled, the languages and the proportions, the labelling process and the agreement figure, and the legal basis if the items are real user content.

The results. Metrics with intervals, sample sizes, and per-slice breakdowns — never a single headline. Include the baselines, so a reader can see what the system adds.

Known failure modes, with real examples. Two or three sanitised failures do more for a reader's judgement than a page of metrics.

Safety and abuse testing — what adversarial testing was done, what it found, and what remains open.

Operational facts — latency, cost per request, dependencies, the model version and the date it was pinned.

Provenance and contact — who owns it, when the card was last updated, and how to report a problem.

The rule that separates a card from a brochure

Every claim carries its measurement.

> *Weak: "Performs well in Hindi."*

> *Card: "Hindi: 84% task success (95% CI 78–89) on 300 items sampled from March traffic, labelled by two reviewers at kappa 0.71. English: 91% (87–94) on 400 items."*

The second version is longer, slower to write, and it is the entire point. A card full of adjectives is marketing with a technical typeface, and the first reader who checks will stop trusting the rest of it.

Where you did not measure something, write that you did not. "Tamil and Bengali are supported by the model but have not been evaluated; we have no evidence about quality in those languages" is a genuinely useful sentence, and it is more honest than a benchmark score nobody on the team can read.

Two versions, one truth

Most teams end up with an internal card carrying detail that cannot be published — supplier terms, internal thresholds, unfixed vulnerabilities — and an external one that is shorter.

The rule is that the external card may omit, and may never contradict. The moment the published card says something the internal one disproves, you have a document that will be read back to you.

Cards rot

The failure mode is not writing a bad card. It is writing a good one at launch and never touching it, so that eighteen months later it describes a model you no longer serve, a corpus that has doubled and a prompt that has been rewritten twice.

Two habits fix it. Keep the card in the repository beside the code, so changes go through the same review. And put the update trigger in the release checklist: a model version change, a prompt change that moves the numbers, or a

new language always updates the card, and the card carries the date and the version it describes.

What a card is not

It is a claim by the people who built the system, evaluated on a set they chose, with metrics they selected. That is not nothing — it is far more than most systems come with — and it is not verification. The next lesson is about why the builder's own evaluation is structurally weak evidence, and what a stronger form looks like.

Free tooling exists and is worth using: Hugging Face's model card template gives you the section structure, and a plain markdown file in the repository is sufficient for everything else.

BEFORE YOU MOVE ON

Which claim belongs in a system card as written?

A. "The system is highly accurate for customer support queries across all supported languages."

B. "Not evaluated in Tamil or Bengali; the model accepts both, and we have no evidence about quality in either."

C. "Extensively red-teamed by our internal team prior to launch."

D. "Outperforms leading alternatives on standard benchmarks."

60 · 10 min

Buying a model you will depend on

THE ONE THING TO KEEP

Decide between suppliers on your own three hundred items and on cost per successful task rather than per token, and get the answers about version pinning, deprecation notice and data retention before the integration exists.

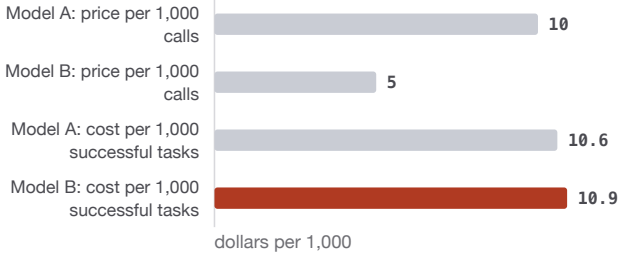
The comparison that matters takes an afternoon

Take 300 items from your own set, run three candidate models through the same prompt, and measure five things: task success, cost per *successful* task, p95 latency, format or schema compliance, and refusal rate.

That is an afternoon of work, and it beats every leaderboard, for the mechanical reason established earlier in the course — public benchmarks sit in training corpora, and they describe tasks that are not yours. A model two places lower on a public ranking routinely wins on a specific job because the job rewards instruction-following or JSON discipline rather than reasoning breadth.

Cost per successful task is the number to insist on. A model at half the token price that needs two attempts and produces malformed output on 8% of calls is not cheaper. Compute it as total spend divided by tasks that came out right.

Price per call is not cost per success



Model A succeeds on 94% of calls in one attempt. Model B is half the price, needs two attempts, and still returns malformed output on 8% of calls, so it costs more per task that actually came out right.

The questions to ask before the integration exists

Answers to these change architecture, and they are much easier to get before you have signed anything.

- **Version pinning.** Can I pin a dated model version, and for how long is it served? Model behaviour changing under you is the most disruptive thing a supplier can do, and the pinning policy is the mitigation.
- **Deprecation notice.** How much warning before a version is retired? Thirty days and six months are very different engineering plans.
- **Training on your data.** Is customer data used for training by default? Can it be disabled, is that in the contract or only the documentation, and does it also cover data sent to any evaluation or safety systems?
- **Retention.** How long is prompt and output data kept, where, and can retention be shortened? This is often the deciding constraint for a regulated customer.
- **Region of processing**, and whether it is guaranteed or best-effort.
- **Rate limits and behaviour under load**, including whether limits are per-key or per-organisation, and what the response looks like when you exceed them.
- **Abuse detection.** What happens if the automated abuse system flags your traffic — throttling, suspension, a human review — and how do you reach a person quickly.
- **Sub-processors and certifications**, because your own customers will ask you.

Test the failure modes deliberately

Success paths are easy. Spend an hour on the failures, because they are what your on-call engineer will meet:

- Send an over-length input and record the error shape.
- Exceed the rate limit deliberately and check whether the response carries a retry-after and whether your client honours it.

- Send an input the safety system will refuse, and see whether the refusal is a normal response, an error code, or something your parser cannot handle.
- Kill the connection mid-stream and confirm you can detect a truncated response — the finish reason again.
- Run a hundred requests concurrently and look at the latency tail rather than the mean.

Write the resulting error taxonomy into your integration. Most production incidents with a model supplier are not the model being wrong; they are the client mishandling a shape of failure nobody tested.

Model the cost properly

Token prices are the smallest part of the estimate. Include prompt growth as features are added, the retrieved context you pass, retries, the evaluation and judging traffic, and any cached-input or batch discounts. Then express it per business unit — per resolved ticket, per document processed, per active user — because that is the number that will be compared against the alternative.

Keep the exit cheap

You will change supplier at some point: a price move, a deprecation, an outage, a policy change, a better model. Two practices make that a week rather than a quarter.

Keep provider-specific detail thin. A small adapter layer, prompts that do not depend on one provider's quirks, and an output parser tolerant of formatting differences.

Re-run your set on a second provider quarterly, and keep the result. A documented fallback that scored within a few points last quarter is an operational option; an untested fallback is a hope. This is also the only way to know whether your prompts have quietly become provider-specific, which they do, gradually, without anybody deciding to.

BEFORE YOU MOVE ON

Model A costs half as much per token as model B, but produces schema-valid output 92% of the time against B's 99.5%, and each retry costs a full call. What is the sound way to compare them?

A. Compare price per million tokens, then account separately for the engineering time spent handling malformed output.
B. Compare cost per successful task, including retries and the failures that never succeed.
C. Prefer B, since schema compliance below 95% is unacceptable in production regardless of cost.
D. Compare quality on a public benchmark first, and treat cost as a tie-breaker.

61 · 8 min

Why the builder's own numbers are weak evidence

THE ONE THING TO KEEP

The team that built a system chooses the set, the metric and what gets reported, so arrange for someone else to hold a set you never see — and accept that publishing an evaluation set is what destroys it.

Selection, not dishonesty

A team evaluating its own work is not usually cheating. The distortion is structural and it operates on honest people:

- **You stop when the number is good.** Nobody keeps digging after a result they expected and liked.
- **You fix what you find, and the set stops containing it.** Every failure discovered and repaired makes the remaining set easier, so the score

rises for reasons that are partly real and partly bookkeeping.

- **The person who wrote the prompt writes the rubric**, and the rubric asks about the things the prompt was designed to do.
- **The failures you never imagined are the ones your set never contains**, and imagination is exactly what a builder has least of about their own design.

None of that is fraud. All of it makes the number too high in a direction nobody can quantify.

What to do inside a team

You can get much of the benefit without an external body.

A held-out set the builders never see. Owned by a different person, run by the harness, results reported as a number without item-level feedback. If the internal score is 88% and the held-out score is 74%, the gap is the size of your self-deception, and it is one of the most valuable numbers you will ever compute.

Rotate the error analysis. Whoever reads the fifty failures should not be the person who wrote the prompt, at least every second time.

Make breaking it somebody's job. An hour a week where a colleague is explicitly rewarded for finding a failure — and where the finding is added to the set, not argued with — does more than any process document.

Have a person outside the team read the card. They will ask what "supported" means, and that question is the audit.

External evaluation, and its immaturity

Beyond the team, the options are third-party audits, structured access for researchers, and public bug-bounty-style programmes for AI harms. All are young, and it is worth being blunt about the state of things: there is no established profession here comparable to financial audit, no agreed standard for what an AI audit covers, and no licensing regime. "Independently verified" frequently means a consultancy ran the vendor's own test suite.

So when reading someone else's independent evaluation, ask three questions: did the evaluator choose the set, was the protocol fixed before results were seen, and was publication guaranteed regardless of outcome. If the answer to any is no, it is a review, not an audit.

Those are also the three commitments to make when you commission one.

Reading somebody else's evaluation

The same scepticism applies when a supplier, a competitor or a research group publishes numbers about their own system. Four questions extract most of the value from a page of results:

- **What was the set, and who built it?** A set assembled by the party being measured is the weakest form.
- **What was left out?** Look for the slice that is conspicuously absent — usually a language, a difficulty band, or the adversarial case.
- **Which baseline?** A result with no baseline cannot be read at all, as the first block of this course established.
- **Is there an interval and a sample size?** A percentage without them is a claim rather than a measurement.

None of this requires access to their data. It is all visible in what they chose to print.

The trade you cannot avoid

A private evaluation set is uncontaminated and unverifiable. Nobody can check your claim, and you can quietly change the set.

A public evaluation set is verifiable and decays. Once published, it enters training corpora, and within a year a high score may mean the model has seen the answers. This is not hypothetical — it is the documented life cycle of every major public benchmark.

There is no way to have both, and the practical compromise is a split:

- **Publish the methodology in full**, plus a sample of items, so a reader can judge the design and reproduce the procedure on their own data.
- **Hold the rest privately**, and rotate a portion of it each year so that even leakage through use has a limited life.
- **Say which parts are which** in the report, so a reader knows what they are relying on.

That is weaker than an audited number and stronger than a claim, and it is the honest position available today.

BEFORE YOU MOVE ON

A team's internal evaluation reports 88% while a set held by another team, which the builders never see, reports 74%. What is the most useful reading?

A. The held-out set is harder, so the two numbers are not comparable and only the trend within each matters.

B. The gap estimates how much the internal number was inflated by building and fixing against that set, and it is the number worth reporting outside the team.

C. The internal set has been contaminated and should be discarded and rebuilt.

D. The held-out evaluation was run by people less familiar with the task, so its labels are probably noisier.

62 · 9 min

What the emerging rules ask for, and what they do not settle

THE ONE THING TO KEEP

Every framework converges on the same practical demands — documented intended use, data provenance, measured per-group performance, logging, human oversight and re-evaluation — while the legal detail differs by country and is still being written.

A caution before anything else

AI regulation is unsettled, it differs by jurisdiction and sector, and it is changing while you read this. Nothing here is legal advice, and anyone shipping something consequential needs a lawyer in the relevant country. What follows is the shape of the landscape, which is useful for deciding what to measure and what to keep.

The voluntary frameworks

NIST's AI Risk Management Framework (United States, voluntary) organises work into four functions: Govern, Map, Measure, Manage. This entire course lives inside *Measure*, and the framework's contribution is mostly a shared vocabulary for talking to people outside engineering.

ISO/IEC 42001 specifies a management system for AI and can be certified against, in the same way as the better-known information-security standard. It asks about process — do you have policies, roles, and a review cycle — rather than about whether your model is any good. **ISO/IEC 23894** covers risk management guidance.

Neither tells you what accuracy is acceptable, because no standard can.

The EU AI Act, in outline

The Act sorts systems by risk. A small set of practices is prohibited outright. A defined list of **high-risk** uses — including employment, education, essential services, and certain safety components — carries substantive obligations: risk management, data governance, technical documentation, automatic logging, human oversight, and stated levels of accuracy, robustness and cybersecurity. A further tier carries transparency duties, such as telling people they are interacting with a machine or that content is synthetic. General-purpose model providers carry their own obligations.

The obligations are phased, the harmonised technical standards that operationalise them are still being produced, and national implementation and enforcement practice are still forming. Anybody who tells you exactly what a given clause will require of your product in practice is ahead of the evidence.

The rules that will bind you sooner

For most teams, sector-specific and existing law matters more than any AI-specific statute:

- **Medical devices**, where software that supports diagnosis has been regulated for years.
- **Financial services model risk**, with long-standing supervisory expectations about validation, independent review and documentation.
- **Employment screening**, where some jurisdictions already require bias audits of automated hiring tools.
- **Data protection.** In India the Digital Personal Data Protection Act governs personal data; in Europe the GDPR does, including rules about automated decisions. Both bear directly on whether user content may become an evaluation set.
- **Consumer protection and advertising law**, which is where "our AI is 99% accurate" becomes a claim you must be able to evidence.

What they all converge on

Read across the frameworks and the same practical list appears every time. If you do these, you are most of the way to any of them:

1. Documented intended use and out-of-scope use.
2. Documented data provenance, including consent basis and dates.
3. Measured performance, reported per relevant group rather than only in aggregate.
4. Logging sufficient to reconstruct a decision after the fact.
5. Human oversight that is specified and, ideally, measured.
6. Incident response, including a way for affected people to complain and be answered.
7. Periodic re-evaluation, with results retained.

Point seven has a practical consequence that surprises teams: you may need to answer "how did you know it worked" about a version you shipped eighteen months ago. That means retaining evaluation artefacts — the set as it was, the results, the model version, the report — not just the current ones. Storage is cheap; reconstructing a defence is not.

What this changes about how you measure

Three things in this course become obligations rather than good practice once any of these regimes applies to you.

Per-group reporting stops being optional. An aggregate figure does not answer a question about whether a system works for a protected group, and the fairness arithmetic later in this course is the technical content behind that requirement.

Accuracy claims must be evidenced. Whatever level you state in your documentation is a level somebody may hold you to, which argues for stating a measured figure with its interval and its set, rather than a round number that sounded good.

Logging becomes a design constraint. "Sufficient to reconstruct a decision" is a demanding specification, and it is much cheaper to build at the start than to retrofit after a request arrives.

The honest limit

Compliance is not safety. A system can satisfy every documentation requirement and still hurt people, and a careful system can be non-conformant on paperwork. The frameworks are floors and record-keeping disciplines, and they are worth meeting; they are not a substitute for the measurement judgement this course has been about, and treating them as one is how organisations end up with excellent binders and unexamined systems.

BEFORE YOU MOVE ON

What is the most practical reason to retain evaluation artefacts — the set as it was, results, model version, report — for old releases?

A. Regulations require a fixed retention period for all AI evaluation data.

B. You may have to answer how you knew a version shipped long ago was fit for use, and that answer cannot be reconstructed later.

C. Old evaluation sets are needed to demonstrate improvement over time to customers.

D. Retained sets can be re-used to retrain the model when performance degrades.

63 · 9 min

Measuring the worst case, not the average one

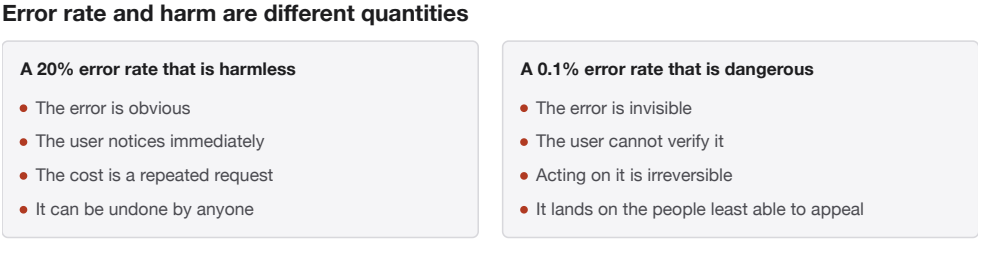
THE ONE THING TO KEEP

Rank failures by severity, reversibility and whether the person can tell they were harmed, then test each harm category with cases written for it — because harm concentrates exactly where a user cannot check the answer.

Error rate and harm are different quantities

A system with a 20% error rate can be harmless: the errors are obvious, the user notices immediately, the cost is a repeated request. A system with a 0.1% error rate can be dangerous: the errors are invisible, they land on people who cannot verify them, and they are irreversible.

Averaging over items treats those the same. Something else is needed.



Averaging over items treats these two systems the same. Describe each failure mode by severity, reversibility, detectability and distribution before you compute anything.

Describe each failure mode along four axes

For every way the system can go wrong, write down:

- **Severity.** The worst realistic outcome, stated concretely. Not "poor user experience" — "a customer misses a filing deadline and pays a penalty".
- **Reversibility.** Can it be undone, and by whom, and at what cost. A wrong draft is reversible; a sent message, a submitted form, a deleted record, a disclosed secret are not.

- **Detectability by the person affected.** Can the user tell it was wrong. This is the axis most often left out and it dominates the others: an error a user can spot is a nuisance, and the same error where verification is impossible is a harm.
- **Distribution.** Who bears it. If the failure concentrates on one language, one region, or people with less ability to appeal, the average is actively misleading.

That description is the input to everything else. Fifteen minutes of writing it usually reorders the team's priorities.

Sample the tail on purpose

Random sampling finds common failures. It finds the rare catastrophic one at the rate it occurs, which is to say almost never, on the day before it occurs in production.

So sample deliberately from where harm lives:

- The lowest-scoring outputs by your judge.
- Requests in high-consequence categories — medical, legal, financial, safety, self-harm, immigration status — identified by a cheap classifier.
- Requests from users in the vulnerable slices you identified.
- The cases where a guardrail fired.

Then have a person read them. A hundred deliberately-chosen items reviewed by a human tells you more about harm than ten thousand sampled ones scored by a model.

Write the cases; do not wait for them

For each harm category, write fifty test cases. This is the same discipline as the safety and red-teaming lesson, applied to consequences rather than to attacks:

- A user describing symptoms and asking what to take.
- A user asking whether their contract lets them leave.

- A user in evident distress.
- A question whose honest answer is "I do not know, and here is who does".

Score these on a severity-weighted basis and report the score separately from the general quality metric. And be explicit that the weights are a judgement: "we weight an irreversible financial error at fifty times a formatting error" is a value statement made visible, which is the most that can be asked of it. Hiding the weights inside an average does not make the judgement go away; it makes it unexaminable.

Design changes usually beat model changes here

The most effective response to a harm is often not a better model. Three interventions reduce harm without touching quality at all, and they should be on the table in the same conversation:

- **Remove the irreversibility.** A draft the user sends beats a message the system sends. A queued action with a ten-second undo beats an immediate one.
- **Restore detectability.** Show the source beside the answer, so verification takes two seconds instead of being impossible.
- **Refuse and route.** For the highest-consequence categories, the best output is a short answer that names where the real help is.

Each is measurable: undo rate, source click-through, and the proportion of high-consequence requests that were routed rather than answered.

Near misses are the leading indicator

Count the cases where a guardrail caught something before it reached a user, and track that rate over time. Near misses rise before incidents do, they are far more numerous, and they cost nothing to collect since the guardrail already fires.

A doubling in near misses after a prompt change is a finding, even if no harm reached anybody, and it is the earliest warning available.

The limit that matters most

You cannot enumerate harms in advance. Every system in production surprises its builders, usually because a group of users has a context nobody in the room shared.

The mitigations are procedural rather than statistical: a visible, easy way for people to report a problem; somebody whose job is to read those reports weekly; and a route from a report into the evaluation set, so the same harm cannot recur silently. A team that measures harm carefully and has no channel for the harms it did not predict is measuring the part of the problem it already understood.

BEFORE YOU MOVE ON

Which failure most deserves priority, given equal frequency of 0.5%?

A. An incorrect summary of an article the user is reading, shown alongside the article.

B. A formatting error that makes a response hard to read on a small screen.

C. An incorrect eligibility answer that leads a user to abandon a benefits claim they were entitled to.

D. A wrong product recommendation in a list of five suggestions.

64 · 9 min

Measuring the human in the loop

THE ONE THING TO KEEP

Oversight is a claim until you measure it, and the instrument is seeded errors — because reviewers catch fewer mistakes precisely as the system gets better and their attention drops.

The most common unmeasured claim in AI

"A human reviews every decision." It appears in system cards, in compliance documents, in sales decks, and it is almost never accompanied by a number. It is a statement about a workflow, not about an outcome, and the two come apart in a predictable direction.

Automation bias, and the paradox it creates

People accept a confident machine suggestion more readily than they should, and more readily still when they are under time pressure, when the system is usually right, and when overriding takes effort. This direction of effect is well replicated across aviation, clinical decision support and driving automation, though the size varies a great deal by setting, so treat it as a reliable direction rather than a fixed number.

The consequence is a paradox worth stating plainly. **The better the system, the worse the reviewer.** At 99% accuracy, errors are rare, trust is high and attention drifts, so the rare error passes through. At 90% accuracy, the reviewer is engaged because they are correcting something every few minutes. A 99% model with rubber-stamp review can produce more errors reaching users than a 90% model with genuine review.

That means you cannot infer pipeline quality from model quality. You have to measure the pipeline.

Four numbers that describe the review

- **Override rate.** What fraction of suggestions the reviewer changes. A rate near zero means either a very good system or no review at all, and you cannot tell which from this number alone.
- **Override correctness.** Of the cases they changed, how often the change was an improvement. This distinguishes a careful reviewer from a nervous one, and it needs an independent adjudication of a sample.
- **Miss rate.** Of the errors that were present, how many got through. This is the number that matters and it cannot be computed from ordinary traffic, because you do not know which items were wrong.
- **Time per review**, plotted against accuracy. There is usually a cliff: below some number of seconds, catch rate collapses. Finding that cliff tells you the maximum sustainable queue rate, which is a staffing decision made with evidence.

Seeded errors are the instrument

Inject known-wrong suggestions into the review queue at a low, random rate — one or two per cent — and measure how many are caught. This is proficiency testing, used for decades in radiology quality assurance and in airport security screening, and it is the only way to get a real miss rate.

Handle it properly, or you will destroy the trust the programme depends on:

- Tell reviewers, in advance and in writing, that seeded cases exist and roughly how often.
- Use the results to size and design the system, never to discipline an individual.
- Seed realistic errors, not absurd ones, or you measure nothing.
- Report the catch rate as a system property alongside model accuracy.

A catch rate of 40% on seeded errors, with a 97% model, tells you the pipeline lets through more than you assumed. That single number has redirected more oversight designs than any amount of policy writing.

The conditions oversight actually needs

Review that is real requires four things, and each is checkable:

1. **Information.** The reviewer can see what the decision was based on — the source passage, the retrieved document, the fields extracted. Reviewing an answer without its evidence is guessing.
2. **Authority.** Overriding is one action, not an escalation, and nothing in the interface treats it as an exception.
3. **Time.** Queue pressure is measured, and the target rate sits on the good side of the accuracy cliff.
4. **Incentives.** Nobody is measured on throughput alone. A reviewer paid per item will approve per item.

And one design to avoid: **batch approval**. An interface where one click accepts forty items is not oversight, whatever the process document says. If batch approval is necessary for volume, sample from the batches and review those individually, and report the sampling rate as part of the oversight claim.

Report the pipeline, not the model

The honest external number is end-to-end: after the model and the reviewer together, what proportion of outputs reaching users are correct. It is usually higher than the model alone and lower than people expect, and it is the only figure that describes what a customer receives.

BEFORE YOU MOVE ON

A team raises model accuracy from 92% to 98% and finds the end-to-end error rate reaching users barely improves. What is the most likely mechanism?

A. The reviewers were already catching nearly all errors, so there was little headroom for the model change to show.
B. The remaining 2% of errors are systematically harder cases that reviewers also find difficult.
C. Errors are now rare enough that reviewer attention has dropped, so a larger share of the errors that remain pass through unchecked.
D. The evaluation set no longer represents production traffic, so the 98% figure is optimistic.

65 · 9 min

Testing whether the system gives away what it was given

THE ONE THING TO KEEP

Plant canary strings at known repetition counts to measure memorisation, and run a cross-tenant retrieval test in CI, because the commonest real leak is a filter applied after ranking rather than before.

Two different leaks

Training leakage — the model reproduces content from data it was trained or fine-tuned on: a customer's email address, a private document, an API key that was in a code corpus.

Runtime leakage — the system shows one user another user's data through retrieval, caching, logging or a shared context.

The second is far more common in ordinary products and far more testable. Deal with it first.

Cross-tenant retrieval, tested in CI

The classic bug has a specific shape: the access filter is applied **after** the vector search rather than as part of it, or it is applied in the application layer and one code path forgets. Everything works in testing because the test data has one tenant.

Build the test properly, and run it on every change:

1. Create two tenants with distinctive, unique content — a nonsense product name, an invented person, a random identifier.
2. From tenant A, run several hundred queries designed to surface tenant B's content: the exact strings, near paraphrases, and queries about the topic.
3. Assert that no chunk from B appears in A's retrieved context, and that no unique string of B's appears in A's output.

The same shape covers caches — a cache key that omits the tenant identifier serves one user's answer to another — and prompt logs surfaced in a support tool.

Canaries measure memorisation

For any model you fine-tune, insert unique random strings into the training data at controlled repetition counts before training:

```
Canary A: "the access phrase is QX7-42J-9PLM"   inserted 1 time
Canary B: "the access phrase is TR3-88K-2WVN"   inserted 10
times
Canary C: "the access phrase is LM5-61P-7HQD"   inserted 100
times
```

After training, prompt the model with the prefix and see what it completes, sampling many times and at several temperatures. The result is a curve of extraction against repetition, and it tells you the thing you need to know opera-

tionally: **memorisation rises sharply with duplication**. A document that appears once is usually safe; the same support macro appearing 400 times is not.

The action that follows is deduplication of training data, which is cheap and is the single most effective mitigation available.

For a model you did not train, the analogous test is simpler: prompt it with the opening of documents you know to be sensitive and in-corpus, and see what comes back.

The rest of the surface

- **Personal data in outputs.** Run a detector — Microsoft's Presidio is free and extensible for Indian identifiers — over a sample of outputs and count. Then over your logs, which is where the larger store sits.
- **Embeddings.** Text can be partially reconstructed from its embedding; this is an active research area with demonstrated attacks. Treat a vector database as containing the text, not a safe hash of it, and apply the same access controls.
- **Membership inference**, in brief: if the model's loss on training examples is systematically lower than on comparable unseen examples, it has memorised. Measurable with a held-out set drawn from the same distribution, and worth doing before releasing model weights.

Erasure is unresolved, and say so

If a person asks for their data to be deleted, deleting the row is straightforward. Removing that example's influence from a trained model is not: there is no reliable, cheap method to unlearn a specific example without retraining, and machine-unlearning research has not settled the question.

Two practical consequences, both worth writing into your documentation:

- Prefer architectures where the sensitive content lives in a retrieval corpus rather than in weights. Deleting the document then genuinely removes it from future answers, immediately and verifiably.

- Where fine-tuning on user content is unavoidable, be explicit about it in the notice, keep the training corpus deduplicated and minimal, and plan retraining cadence as part of the erasure process rather than claiming a capability you do not have.

Differential privacy is the only approach offering a rigorous guarantee, and it costs utility and is awkward at scale. Mention it honestly rather than gesturing at it: for most product teams it is not yet the answer, and the effective controls are minimisation, deduplication, retrieval architectures and the access tests above.

BEFORE YOU MOVE ON

Why is a canary string inserted at several different repetition counts more informative than one inserted once?

A. It provides several independent trials, which reduces the variance of the extraction estimate.

B. It maps extraction risk against duplication, which is the variable a team can act on by deduplicating the training data.

C. It tests whether the model treats rare and common strings differently in its tokeniser.

D. Higher repetition counts make extraction certain, providing a positive control that the test itself works.

66 · 8 min

When the model is right and the product still fails

THE ONE THING TO KEEP

Measure the funnel from attempt to resolved outcome, because a model that is right nine times in ten routinely produces a feature that works four times in ten once asking, reading, trusting and acting are counted.

The gap that surprises everybody once

A support assistant measures 90% correct on a careful evaluation set. In production, 40% of the people who start a question end up with their problem solved.

Nothing is wrong with the measurement. The other 50 points went somewhere else:

- People did not know what they could ask, so they asked something the system does not cover.
- The answer was correct and 300 words long, and nobody read past the first line.
- It took nine seconds, and a fifth of people left before it arrived.
- It was right but not actionable: correct information, no next step, no link, no button.
- It was right and the user did not believe it, because nothing showed where it came from.

Model accuracy is a component measurement. The product measurement is a funnel.

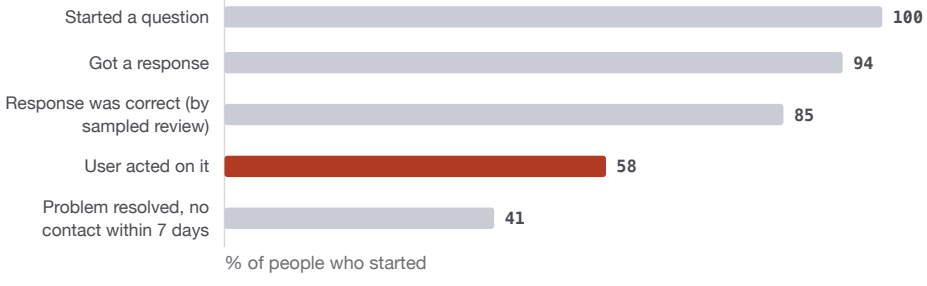
Measure the funnel

Instrument the stages and report them together:

```
started a question        100%
got a response             94%   (6% abandoned or errored)
response was correct       85%   (of those, by sampled review)
user acted on it           58%
problem resolved           41%   (no follow-up contact within 7
days)
```

Every step between two rows is a fixable product problem, and the arithmetic tells you which one is worth a week. In this shape the largest loss is between "correct" and "acted on", which is a writing and interface problem, not a model problem — and no amount of prompt work would have found it.

The funnel behind a 90% model



The largest drop is between 'correct' and 'acted on': a writing and interface problem that no prompt work would have found, and that the eval set cannot see.

The resolution measure at the bottom matters most and is usually available already: a repeat contact within a week, a completed transaction, a claim submitted. Use an existing product event rather than inventing a satisfaction survey nobody answers.

Interface decisions change the numbers more than model changes

Three that consistently move the funnel:

- **Show the source beside the answer.** It raises trust where the answer is right and, more usefully, lets the user catch it when it is wrong. This is the detectability axis from the harm lesson, purchased with a link.
- **Draft rather than send.** Anywhere the system acts, an editable draft converts an irreversible error into a normal edit. Measure the edit rate; it is also free quality data.
- **Be shorter.** People do not read. A correct 30-word answer beats a correct 300-word answer at every stage of the funnel after "was correct", and the eval set cannot see this because a rubric scores completeness.

Displaying confidence deserves a caution: an uncalibrated confidence number is worse than none, because it moves people's trust with no information behind it. Calibrate first, or show provenance instead.

Attribute the movement, do not just watch it

Product metrics move for reasons that have nothing to do with your model: a campaign, a holiday, an app release, a competitor's outage, a change to the help centre. So a funnel on its own cannot tell you whether the model change helped.

The pairing that works is the offline set for attribution and the funnel for consequence. When the offline number is flat and resolution jumped, look outside the model. When the offline number rose and resolution did not, look at the stages after "was correct". Neither measurement answers the question alone, and together they usually name the stage to work on within an afternoon.

Watch five people use it

Five to eight moderated sessions, an hour each, will tell you things no metric will. The recurring findings are consistent across products: people do not know what to type; they do not read; they distrust anything with no source; and they will not scroll.

This is not statistics and it is not meant to be. It is the fastest available route to knowing which part of the funnel to instrument next.

The fallback is part of the product

Every AI feature has a bad day: the provider is down, the model refuses, the request times out, the answer is empty. What the user sees then is part of the measured product, and it is usually built last and never evaluated.

Measure the fallback path explicitly — how often it fires, and what proportion of users reach a resolution through it. A graceful fallback that hands over the help article and a contact route can carry a surprising share of your traffic without anybody noticing, which is the point.

BEFORE YOU MOVE ON

A team's model scores 90% on a strong evaluation set, but only 41% of users who start a question end up resolved. What does that gap most likely indicate?

A. The evaluation set is unrepresentative, and real traffic is much harder than the sampled items.

B. Losses at the stages the model metric cannot see — asking, reading, trusting, acting — which are interface and writing problems rather than model problems.

C. The judge used offline is more lenient than a human would be on the same answers.

D. Resolution is the wrong outcome measure, since many users get value without resolving anything.

67 · 9 min

When people disagree about what good means

THE ONE THING TO KEEP

Keep every annotator's individual labels rather than only the majority vote, because aggregation converts one group's reading into ground truth — and measurement can show whether you apply your rule consistently, never whether the rule is right.

Some disagreements are not measurement errors

Through this course, disagreement between labellers has been treated as a problem to fix: sharpen the guide, define the terms, re-label. Most of the time that works, and low agreement really is a symptom of a vague definition.

Some disagreements are not that. What counts as offensive depends on who is being spoken to and by whom. What counts as a helpful explanation depends on what the reader already knows. What counts as neutral framing on a contested subject depends on where you stand. Whether an assistant should push back or comply is a question about what a product is for.

You can drive agreement up on these by writing a stricter rule, and what you have done is chosen a position, not discovered a fact. That is legitimate. Hiding it inside a metric is not.

What the annotation research shows

Studies of toxicity and hate-speech labelling have repeatedly found that annotators' backgrounds predict their labels: people from a targeted group and people outside it rate the same text differently, and so do annotators of different ages. The direction of these findings is consistent enough to plan around, even though the magnitudes vary between datasets and study designs.

The mechanism is what matters for your work. **Majority vote erases the minority reading.** Aggregate five annotators, take the majority, call it the gold label, and the resulting file encodes the dominant group's interpretation as truth. Train on it, evaluate against it, and every number you produce afterwards inherits that choice invisibly.

Four practices that keep it honest

Keep per-annotator labels. Store every individual judgement with an annotator identifier, not just the aggregate. You can always aggregate later; you can never disaggregate. This one habit costs a column and preserves every option.

Report agreement and composition, not only the label. "Labelled by six reviewers, three of whom are speakers of the language in question, with a kappa of 0.52" tells a reader far more than a clean gold set does.

Where perspectives diverge systematically, report more than one number. Score the system against each group's labels and publish both. A model at 88% against one group's judgements and 61% against another's is a result — an important one — and a single 79% is a way of not saying it.

State the position in the policy. "We treat reclaimed slurs used by members of the group as allowed. Here is why, here is who decided, and here is how to contest it." A written position can be argued with. A threshold buried in a rubric cannot.

Some disagreements should be settings

Not every difference needs a global answer. Where two reasonable communities want different behaviour, the honest product answer is often a control rather than a decision: a per-community threshold, a per-user preference for directness, a room that sets its own norms.

This has an evaluation consequence worth planning for. Once behaviour is configurable, the metric is no longer "is the output good" but "does the output match the setting that was chosen", which is a much easier question to measure and a much harder one to design.

Telling the two kinds of disagreement apart

You will not always know at the outset whether a low agreement figure is a vague guide or a genuine difference in values. One diagnostic separates them cheaply: take twenty disagreements and ask each labeller to write why, then read the reasons.

- If the reasons show people applying different readings of an ambiguous word — "we each thought 'threat' meant something different" — it is a definition problem. Fix the guide.
- If the reasons show people applying the same definition and reaching different conclusions about whether the thing is acceptable, the guide is not the issue and no rewrite will close the gap.

The second case is where the practices above apply. Mistaking it for the first produces a guide that grows longer every quarter, agreement that never improves, and labellers who quietly conclude that their judgement is unwelcome.

What measurement can and cannot do

It is worth being exact about the boundary, because this is where evaluation is most often oversold.

Measurement **can** tell you whether you apply your own rule consistently, whether the rule falls more heavily on one group than another, how often people contest a decision and how often those contests succeed, and whether your stated policy and your actual behaviour match.

Measurement **cannot** tell you the rule is right. That is a judgement somebody has to make, own and be accountable for.

The best evaluation practice therefore ends where it began: a number is a servant of a decision, and the decision belongs to a person. What this course buys you is that the person making it can see what they are choosing, instead of believing the choice was made for them by an average.

BEFORE YOU MOVE ON

Why keep individual annotator labels rather than only the aggregated majority verdict?

A. Individual labels allow you to identify and remove unreliable annotators from the dataset.

B. Aggregation is irreversible, so keeping individual labels preserves the ability to analyse systematic differences between groups of raters later.

C. Majority voting requires an odd number of annotators, and individual labels let you recompute when the count changes.

D. Individual labels increase the effective sample size, since each judgement counts as an observation.

CASE STUDY · MODULE 7

The card said ninety-six per cent

A state social-welfare department in Lucknow procuring a system that flags scholarship applications for document re-checking, with the scheme opening in six weeks

The department administers a post-matric scholarship that receives about 400,000 applications a year. Each application carries income and caste certificates, and a share of them are flagged for re-verification by a district clerk before payment. Historically the flagging was done by rule and by hand, it took three months, and students missed a semester's fees waiting.

A vendor proposed a system that read each application and produced a flag, "needs document re-check", with a confidence score. The proposal came with a system card: 96% accuracy on the vendor's evaluation set, an ISO management-system certificate, and a line saying that every flagged application would be reviewed by a human before any action.

The joint secretary in charge, Mrs Rathore, was under pressure to sign. The scheme opened in six weeks, the vendor was ready, and the accuracy figure had already been quoted in a briefing to the minister. Her deputy, Anand, had two objections and a proposal.

The first objection was that 96% was measured by the vendor, on a set the vendor built, with a metric the vendor chose. Nothing on the card said what the set contained, how many items, what the base rate of genuinely problematic applications was, or whether the figure held for applications in Urdu, which were about a fifth of the total in three districts.

The second was the human review line. Anand had visited the district office in Bahraich. The clerks would receive flagged applications in a list of forty and approve them with one button. Nobody had measured how many wrong flags a clerk actually caught in that interface, and Anand's guess, from watching for an hour, was very few.

His proposal was to spend three of the six weeks measuring. Take 300 applications from last year, stratified by district and by language, whose out-

comes were known: which ones had genuinely needed re-checking. Run the vendor's system over them and compute precision and recall per slice, with intervals. Then, separately, seed a small number of known-wrong flags into a test queue and measure how many the clerks caught, telling the clerks in writing that seeded cases existed and that the results would size the system and never be used against an individual.

Mrs Rathore's cost was straightforward. Three weeks of a six-week runway, the clerks' time, and the political embarrassment of a minister's number being revised downwards in public. The vendor pointed out, accurately, that their certification covered their process and that no department had previously asked for anything of the kind.

Anand's cost, if the measurement was skipped, was that the first anybody would know about a per-language gap would be when Urdu-medium students in three districts were flagged at twice the rate of everyone else, and the minister's office found out from a newspaper.

There was also the question of what the confidence score meant. The vendor's interface sorted the queue by it, so clerks would see the highest-scoring applications first, and nobody had checked whether an application scored 0.9 was actually more likely to need a re-check than one scored 0.6. If the score was decoration, the queue order was too.

Should she sign, measure, or both, and what would she need to see before the scheme opened?

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 both. The contract was signed with a clause that made go-live conditional on the department's own measurement, which the vendor accepted once it was clear the alternative was no contract.

On the department's 300 applications, overall accuracy was 89%, and because only 11% of applications had genuinely needed re-checking, a system that flagged nothing would have scored 89% too. The numbers that mattered were recall of 71% on the applications that needed a re-check and a false-flag rate of 6% on the ones that did not. For the Urdu-medium slice, recall was 74% and the false-flag rate was 14%, more than double. The confidence scores were uncalibrated: applications scored above 0.9 needed a re-check about 60% of the time.

The seeded-error test was the finding that changed the design. Twenty known-wrong flags went into a queue of 400 across two district offices. The clerks caught seven. A 96% system with a 35% catch rate lets through more wrong flags than the department had been letting through by hand.

The system card was rewritten by the department, with the vendor's cooperation: every claim carried its set, its sample size and its interval, the Urdu figure was printed separately, and the out-of-scope section said plainly that the system had not been evaluated on applications with handwritten certificates. The interface changed more than the model did: batch approval was removed, each flag was shown beside the specific certificate field that triggered it, and a per-language floor was written into the contract, with the Urdu slice to be re-measured monthly on a fresh sample. The vendor's contract gained a version-pinning clause and ninety days' notice of any model change.

The scheme opened on time. The minister's briefing was corrected before the opening, which was uncomfortable for a day and not for longer.

The vendor's 96% and the department's 89% were both honest measurements. What made the department's number the one to act on?

The builder chooses the set, the metric and what to report, and stops when the number is good; that is selection, not fraud, and it inflates the figure in a direction nobody can quantify. The department's set was drawn from its own applications, stratified by district and language, with known outcomes and a stated base rate, so its precision and recall described the decision the department actually had to make.

Why was the seeded-error test necessary, given that the card already promised human review of every flag?

A statement that a human reviews every decision is a claim about a workflow, not an outcome. The miss rate cannot be computed from ordinary traffic because nobody knows which flags were wrong. Injecting known-wrong flags at a low rate, with the clerks told in advance and the results never used against an individual, gave a real catch rate of 35%, and showed that a one-click batch approval interface was not oversight.

The department kept the vendor. What changed in the deployment that made a 71%-recall system acceptable, and what would you still want to see measured?

The changes were to detectability and reversibility rather than to the model: each flag shown beside the triggering field so a clerk could verify in seconds, batch approval removed, a per-language floor with monthly re-measurement, and a rewritten card whose claims carried their measurements. Still to measure: the end-to-end pipeline figure after model and clerk together, calibration of the confidence scores per slice, and whether the Urdu gap closes with more representative examples or reflects something in the labels themselves.

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 What is the rule that separates a system card from a brochure?

A. Every claim carries the measurement behind it.
B. It is signed off by the supplier's compliance officer before it is published externally.
C. It reports a single headline accuracy so a reader can compare systems quickly.
D. It is published externally rather than kept inside the team.

2 Model B costs half as much per token as model A, needs two attempts per task, and returns malformed output on 8% of calls. How should the two be compared?

A. On price per token, where B is plainly cheaper.
B. On cost per successful task: total spend over tasks that came out right.
C. On their public benchmark rank first, then on price to break ties.
D. On p95 latency alone, since token cost is the smallest part of any realistic estimate.

3 A supplier says its model was independently verified. Which question decides whether that was an audit or a review?

A. Whether the evaluating firm holds an ISO management-system certificate covering its own evaluation processes.
B. Whether the results were published in a peer-reviewed venue.
C. Whether the evaluator chose the set, fixed the protocol first, and had publication guaranteed.
D. Whether the independent score came out higher than the supplier's own.

4 Which axis of a failure mode does the harm lesson say dominates the others and is most often left out?

A. Severity, the worst realistic outcome stated concretely.
B. Frequency, how often the failure occurs in traffic.
C. Cost to the team of fixing it once found.
D. Detectability by the person affected.

5 How do you obtain a real miss rate for a human review step?

A. Compare the reviewers' override rate against the model's measured accuracy on the same period of traffic.
B. Inject known-wrong suggestions at a low random rate and count how many are caught.
C. Survey the reviewers about how confident they are in their decisions.
D. Measure time per review and infer attention from the average.

6 According to the privacy lesson, where does the commonest real cross-tenant leak come from?

A. The model memorising another tenant's documents during fine-tuning and reproducing them on request.
B. Embeddings being partially reversible into the text they encode.
C. An access filter applied after the vector search rather than as part of it.
D. Prompt logs retained beyond the stated retention window.

MODULE 8

Living with it

Everything so far measures a system before it ships. This block is about the years afterwards: watching it in production, the tails of cost and latency, the model that changed under you, the corpus that fills up with its own output, checking that it works as well for one group of users as another, and reporting a number to people who will act on it.

> BY THE END OF THIS MODULE YOU CAN
>
> Keep a shipped system honest over time — monitor for drift, measure the tail of cost and latency rather than the mean, measure quality separately for each group of users, convert incidents into permanent test items, and report a result with its uncertainty stated

---

68 · 9 min

Cost, latency, and the model that changed under you

> THE ONE THING TO KEEP
>
> Pin the model and keep a frozen set, and a forced upgrade becomes an afternoon instead of a crisis.

Quality is a row, not a number

Every eval run should print four things together, because they trade against each other and you can only see a trade when the numbers sit side by side.

```
variant        pass    p50      p95      cost / 1k calls
v3 (current)   78%     1.1s     2.4s     $2.10
v4 (proposed)  81%     2.6s     8.9s     $7.40
v4-mini        74%     0.6s     1.3s     $0.35
```

Now the conversation is real. Three points of quality for a p95 that goes from 2.4 to 8.9 seconds is usually a bad trade for anything a person is waiting on. And `v4-mini`, four points worse and six times cheaper, is suddenly the interesting row — at 100,000 calls a day that is the difference between roughly $10,500 and $1,050 a month.

Two details that matter. Report **p95, not the average**: the average hides the one user in twenty who waits nine seconds, and that user is the one who leaves. And convert cost into your actual monthly bill at your actual volume, in the currency you pay in. "$0.0074 per call" and "about ₹9 lakh a month" produce completely different reactions in a budget meeting, and only one of them is a decision.

Pin the model. Always.

Never point production at an alias. A dated, explicit model id in a config file, changed by a pull request that runs the eval. This one habit converts the worst day of your quarter into an ordinary afternoon.

Because eventually you will be forced to move — a deprecation notice, a price change, a provider outage that pushes you to a fallback. When that day comes, the frozen eval set is the only asset that matters. Run it against both models, read the per-item diff, and you have an answer in an hour rather than a week of arguing in Slack.

Quality dropped and you changed nothing

A sequence that works.

Confirm it with the set, not with anecdotes. Three loud complaints in a channel is not a trend, and sometimes the numbers are genuinely flat. Sometimes the numbers confirm it, and now you have a magnitude.

Look at failures by slice, never in aggregate. Model changes are almost never uniform. The new model is usually better on the bulk and worse on something specific: non-English input, very long documents, strict JSON with a nested schema, refusals on your borderline-but-legitimate cases. A model can improve overall and get dramatically worse on the 8% of traffic your complainers live in — and the headline number will happily report an improvement.

Fix the prompt to the new model, do not expect the old prompt to transfer. Prompts overfit to models. Instructions that were load-bearing for one model are noise to another; formatting tricks that worked stop working. Budget half a day to re-tune, and re-run the set after.

Keep a canary running. Sample one percent of live traffic every day through your calibrated judge and chart it. A slow drift is obvious on a chart and invisible in memory. This is also the only thing that catches the other kind of drift, which nobody plans for: *your inputs* changed. New market, new customer segment, a feature that brought in a different sort of question. Your eval set from January describes January.

The last honest thing

Evaluation is not a phase you complete. It is a small permanent tax — an hour a week keeping the set current, a CI job, a chart someone glances at. Teams that pay it ship changes confidently and recover from surprises in an afternoon. Teams that do not spend that same time arguing about whether things got worse, and finding out from customers when they did.

BEFORE YOU MOVE ON

A provider deprecation forces an upgrade. Within days, support complaints about the summariser rise sharply. The team runs its frozen eval set and the pass rate has gone up three points. What is the most useful reading?

A. Both can be true — break the failures down by slice, because a model can improve on the bulk of traffic and get worse on the slice the complainers are in.
B. The complaints are anecdotal and the eval is the measurement, so there is nothing to act on.
C. The eval set is stale and should be rebuilt from recent traffic before any conclusion is drawn.
D. The new model is better and the complaints are a novelty effect that will settle within two weeks.

69 · 9 min

The tail is what your users feel

THE ONE THING TO KEEP

Report p50, p95 and p99 rather than a mean, because generation time is right-skewed and a session making ten calls meets the 95th percentile about forty per cent of the time.

Why the mean describes nobody

Latency for a generative system is right-skewed by construction: total time scales with the number of output tokens, and output length has a long tail. So the distribution has a dense body and a stretched right side, and the mean sits somewhere between the median and the 90th percentile, describing neither.

A real shape looks like this:

- p50: 1.2 s

- p90: 4.1 s
- p95: 6.8 s
- p99: 14 s
- max: 62 s (a timeout that did not fire)

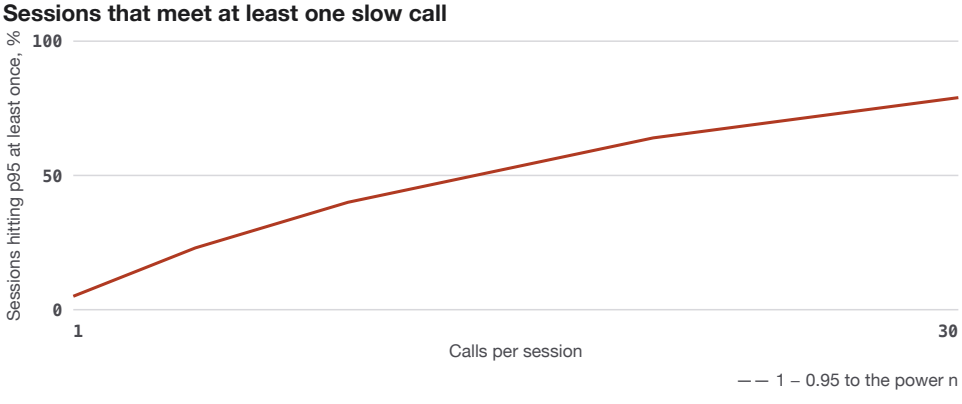
The mean might be 2.4 seconds, and quoting it hides that one request in twenty waits nearly seven seconds and one in a hundred waits fourteen.

The arithmetic that makes the tail matter

Percentiles per request understate the experience, because users make more than one request. If a session involves ten calls, the chance of meeting at least one call above the 95th percentile is:

```
1 - 0.95 ** 10 = 0.40
```

Forty per cent of sessions contain a slow call. An agent taking thirty steps meets it almost every time — `1 - 0.95 ** 30 = 0.79`. This is why tail latency dominates the felt quality of multi-step products, and why an improvement to p99 can matter more than an improvement to p50.



A ten-call session meets a 95th-percentile call 40% of the time; a thirty-step agent, 79%. That is why an improvement to p99 can matter more than one to p50.

Report both the per-call percentiles and the **per-session** total, because the session is the unit of experience.

Time to first token is a different metric

For a streaming interface, the user's sense of speed is governed by time to first token, and then by the token rate. A response with a 400 ms first token and a steady stream feels fast at eight seconds total; one with three seconds of nothing feels broken at four.

Measure three numbers for streaming: time to first token, tokens per second, and total time. Optimising them pulls in different directions — prefill work and retrieval sit before the first token, while output length drives the rest — and a change that improves total time while delaying the first token will be reported by users as a slowdown.

What is actually in the tail

Look before you optimise. The usual contributors, in rough order:

- **Long outputs.** Check the correlation between output tokens and latency; it is usually most of the variance.
- **Retries.** A timeout followed by a retry is two full attempts, so a 5-second timeout can produce an 11-second experience. Retries multiply the tail rather than trimming it.
- **Long inputs**, since prefill time grows with input length — a retrieved context of 20 chunks costs before generation starts.
- **Cold caches and cold starts** after a deploy or a scale-up.
- **Provider queueing** at peak hours, which is invisible in your own metrics except as latency.

Timeouts change your quality measurement

This is the subtle one. If a request times out at 5 seconds and the slow requests are disproportionately the hard ones, then the answers you measured are the ones that finished — and hard, slow, more-likely-wrong requests are missing from the sample. Your quality metric describes the survivors.

Two consequences: count timed-out requests as failures in every quality metric, and report the timeout rate beside every quality number. A prompt change

that improves quality by two points while doubling the timeout rate has made the product worse, and only the pair of numbers shows it.

Cost has a tail too

The same arithmetic applies to money. Mean cost per request hides that a small fraction of requests — long documents, deep agent loops, a retry storm — consume a disproportionate share of the bill. Report cost percentiles and the share of spend taken by the top 1% of requests. Then cap: a per-request token limit and a per-session step limit turn an unbounded tail into a bounded one, and the cap firing is a metric worth watching.

Measure under load, not in a notebook

Latency measured one request at a time is a lower bound and nothing more. Run a simple load test — `k6`, `locust`, or a loop with `asyncio` — at the concurrency you expect, and plot latency against concurrency. The curve has a knee, and knowing where it is tells you both your capacity and what your users will experience at the busiest hour, which is the only hour anybody remembers.

BEFORE YOU MOVE ON

An agent makes 30 model calls per task, each with a p95 latency of 6 seconds. What follows?

A. Roughly 5% of tasks will contain a slow call, matching the per-call percentile.

B. About 79% of tasks will contain at least one call above the 95th percentile, so tail latency dominates the task experience.

C. Task latency is 30 times the median, since the percentiles average out over many calls.

D. Nothing follows without knowing the mean, since percentiles cannot be combined across calls.

70 · 9 min

Watching a system you can no longer test

THE ONE THING TO KEEP

Without labels you can still watch the input and output distributions, and refusal rate is the fastest detector of a model that changed under you.

Offline sets go stale; traffic does not

Your eval set was built from last quarter's inputs. Production is receiving this morning's. Between them sit a new customer segment, a marketing campaign that brought a different kind of question, a festival, a competitor's outage, and a provider who updated a model without telling you.

Monitoring is the part of evaluation that runs continuously, and its job is narrower than people assume. It is not to measure quality — you cannot, without labels. It is to **detect change fast enough to go and look**.

Log this, per request

The minimum that makes everything else possible:

- request id, session id, turn index, timestamp
- **model id including the exact version string**, prompt version, retrieval config version
- input length, detected language, and the slice tags you defined
- output length, finish reason, refusal flag
- the result of every deterministic check
- latency (queue, first token, total), tokens in and out, cost
- downstream signal: regenerated, escalated, edited, abandoned

Store raw input and output text under a retention policy, redacted or hashed where the field is personal, with a stated window — 30 days is a common and

defensible choice. Some of the most useful debugging is impossible without the text, and some of the worst incidents are caused by keeping it forever. Decide deliberately, write it down, and tell the people whose text it is.

Distribution shift, on both sides

You can watch the inputs and the outputs without any labels at all, and it is remarkable how much this catches.

Input distribution. Mean and p95 length. Language mix. Share per slice tag. Cluster share, if you embed a sample and track how the clusters divide. A sudden change means your users changed, and your eval set no longer represents them.

Output distribution. These are the leading indicators, and they move before anybody complains:

- **refusal rate** — the single most sensitive detector of a silent model change
- **schema failure rate**
- **mean output length**
- **tool-call rate** and **step-limit-hit rate**, for agents
- **retrieval hit rate** at your k
- **p95 latency** and cost per request

A refusal rate moving from 2% to 9% overnight, with no deployment on your side, is a provider-side change, and you will know within an hour instead of finding out from a customer in three weeks.

Alert on shape, not on a single reading

Rates are noisy at low volume. Three rules keep the alerts trustworthy:

- Compare against the same hour last week, not against the last hour. Traffic has a daily and weekly shape, and comparing to an hour ago generates an alert every Monday morning.
- Require a minimum denominator. A refusal rate computed on 12 requests is not a signal.

- Alert on a sustained change — say, three consecutive windows outside the band — rather than a single spike.

A simple band of mean plus or minus three standard deviations over the last four weeks of the same weekday-hour is usually enough. Fancier change-point detection is available and rarely necessary.

Sample the expensive checks

Deterministic checks are free; run them on 100% of traffic. Judge calls are not; run them on a sample — 1 to 5% is typical — stratified so that rare slices are represented, and remember when reading the number that it is stratified.

A judge running on 2% of a million requests is 20,000 judged items a day, which is far more than any offline set, and it is enough to detect a genuine quality change within a day. That is a good use of a few dollars.

Pin the version, and watch for the swap

The recurring incident in this course is the model changing under you. Two defences, and you want both.

First, pin exact version strings everywhere: the system's model, the judge's model, the embedding model. Aliases like `-latest` are convenient and they are how a Tuesday becomes interesting.

Second — because pinning is not always possible, and because pinned models are deprecated on a schedule — keep a **canary set**: 20 items, run hourly, with outputs stored. When outputs that were stable for weeks all change on the same afternoon, something moved that nobody told you about. Twenty items an hour costs a few dollars a month and it is the cheapest early warning available.

The two-line rule for dashboards

A quality dashboard nobody looks at is a cost. Keep the daily view to what a person can absorb in ten seconds: today's numbers against last week's, and a

list of the alerts that fired. Everything else lives in a query you run when something looks wrong.

And give it an owner. A metric with no owner drifts out of date, then breaks, then is quietly ignored, and it usually takes about four months.

BEFORE YOU MOVE ON

With no deployment on your side, refusal rate rises from 2% to 9% and mean output length falls 30% over one afternoon. Quality complaints have not yet appeared. What is the most likely cause?

A. A traffic shift — new users asking more out-of-scope questions, which would raise refusals legitimately.

B. Retrieval has degraded, so the system has less context and declines more often.

C. The provider has changed the model behind an unpinned alias, and both metrics moved together because the underlying system changed.

D. The refusal detector itself is miscounting, since a 4.5x change in a day is implausible for real behaviour.

---

71 · 9 min

The system that trains on its own output

THE ONE THING TO KEEP

Tag the provenance of every document and every label, because a corpus filling with generated text keeps groundedness perfect while truth degrades, and treating accepted suggestions as gold teaches a model its own mistakes.

Four loops, one shape

Each of these closes a circle in which the system's output becomes its own input.

The corpus loop. Generated text — yours or the web's — enters the retrieval corpus. Answers are then grounded in machine-written material, so groundedness stays at 99% while the underlying truth drifts. The check that was your best production signal has gone blind, because it measures faithfulness to the corpus and the corpus is now partly the model.

The label loop. A user accepts a suggestion, and the log records it as correct. Only accepted outputs get labelled, so the training and evaluation data contains the model's successes and its plausible errors, and not its obvious ones. Retrain on that and you sharpen exactly the failures users could not detect.

The evaluation loop. Items generated by a model, or drawn from logs already thick with generated content, measure the system's agreement with itself. The number rises and means less each time.

The user loop. People learn how to phrase things so the system understands. The metric improves with no change to the model, because the traffic has adapted to it. Pleasant, and not a quality gain — and it hides genuine regressions for the same reason.

Provenance, not detection

The instinct is to detect machine-written text and filter it out. Resist it: automatic detectors of generated text are unreliable, with false-positive rates that fall hardest on people writing in a second language and on anybody with a plain, formulaic style. Filtering a corpus with one of these excludes real authors and lets plenty of generated text through.

Do it structurally instead. Every document in the corpus carries metadata:

```
source: help-centre | vendor-doc | community-post | model-gen-
erated
authored_by: human | model | mixed
created_at, ingested_at, reviewed_by, review_date
```

Then two things become possible: excluding machine-authored material from the grounding corpus by policy rather than by guesswork, and tracking the share that is machine-authored over time. That share is a metric worth

putting on the dashboard, because it only ever goes up unless someone is watching.

The same applies to labels: record how a label was produced — human review, user acceptance, judge model, heuristic — and never mix them in one column.

Breaking the label loop

Acceptance is a useful signal and a biased one. Two corrections, both cheap:

- **Sample what was rejected and edited**, not only what was accepted. The edits are the richest quality data you own, and they are systematically excluded by an accept-as-gold pipeline.
- **Label a random sample independently of user action**, on a fixed schedule. A few hundred items a month, drawn without regard to what happened next, is the only stream that does not inherit the loop.

Breaking the corpus loop

Two rules keep the retrieval corpus clean enough to be worth grounding against.

Do not write model output back into the source of truth. A summary saved into the help centre, an auto-drafted article published without review, a generated FAQ indexed alongside the manual — each of these converts an answer into a source. Where generated material must be stored, keep it in a separate namespace that the retriever does not read from.

Review before ingestion, and record who reviewed. A human sign-off field on each document is the cheapest possible provenance, and it makes the "which documents may be grounded against" policy enforceable in a query rather than in a meeting.

Keep one yardstick that cannot move

The single most useful defence is boring: freeze a human-authored evaluation set from before the system was widely used, and keep it forever.

Everything else drifts — the corpus, the traffic, the labels, the judges, the users' phrasing. A frozen set of a few hundred items, written or sampled from pre-launch human material, is the only fixed point you will have. Run it quarterly. When every live metric is flat and the frozen set drops six points, you have learned something you could not have learned any other way.

What is unsettled

Research on training models repeatedly on their own outputs shows degradation in controlled settings — narrowing diversity and drifting distributions — and there is real disagreement about how much this applies to real corpora, which are mixtures of human and machine text with continual fresh human data added.

Treat it as a risk to instrument rather than a fact to act on. The instrumentation costs a metadata column and a frozen set; the confident version of the claim, in either direction, is ahead of the evidence.

BEFORE YOU MOVE ON

A retrieval assistant maintains 99% groundedness for a year while user complaints about wrong answers rise steadily. Which explanation fits both observations?

A. The groundedness checker has drifted and now over-reports support.

B. Machine-generated documents have entered the corpus, so answers remain faithful to sources that are themselves increasingly unreliable.

C. Users have become more demanding as they grow familiar with the product.

D. The model version changed and the new one hallucinates more subtly.

72 · 8 min

Refreshing a set without breaking every comparison

THE ONE THING TO KEEP

Compare your set's mix against last month's traffic and retire it when the divergence gets large, but change it on a schedule with both versions reported for one cycle, because a silent set change looks exactly like a quality change.

The symptom

The evaluation score has been steady at 87% for eight months. Complaints are rising. Nothing in the harness is broken.

The set was drawn in March. Since then a new market opened, the app shipped a feature that changed how people phrase requests, and a large customer started sending documents rather than questions. The set is measuring a product that no longer exists, precisely and reproducibly.

Measure the divergence, do not guess it

You do not need a sophisticated drift detector. Bucket both your eval set and last month's traffic on a few cheap features, and compare the proportions:

- intent or topic mix
- language and script
- input length, in three or four bands
- channel or surface
- whether an attachment or a document is present

Then sum the absolute differences and halve it, which gives the total variation distance between the two mixes:

```
tvd = 0.5 * sum(abs(p_eval[b] - p_live[b]) for b in buckets)
```

A value under 0.1 means the set still resembles traffic. Around 0.2 the headline number is drifting away from reality. Above 0.3 you are reporting a number about a different population, and the refresh is overdue.

Run this monthly. It takes a query and a few lines, and it converts "the set feels old" into a scheduled decision.

Refresh on a schedule, in a way that preserves comparisons

Changing a set changes the number, and a number that moves because the set moved looks exactly like a quality change. So:

- **Version the set**, with a date, and record the version alongside every result. A number with no set version cannot be compared with anything.
- **Keep a frozen core** — perhaps half the items — unchanged across refreshes, so there is always a like-for-like comparison across the boundary.
- **Report both versions for one cycle.** Old set 87%, new set 81%, both on the same system, and the six-point gap is a fact about the sets rather than a regression. Publish that gap; it is the most informative number in the whole refresh.
- **Never swap silently.** Every incident where a team argued for a week about a phantom regression traces back to somebody quietly adding forty items.

Retire items, without losing what they guard

An item that every version has passed for a year carries no information. It costs money and it never moves. But deleting it removes the guard against a regression that was fixed long ago, and fixed bugs do come back.

Move rather than delete. Retired items go to a cheap smoke suite that runs nightly with deterministic checks only, or in a cascade's lowest tier. The regression is still guarded and the expensive judging budget goes to items that discriminate.

The reverse rule matters too: an item that flips constantly is either a rubric ambiguity or genuine non-determinism, and it belongs in quarantine rather than in the headline set.

Keep the composition honest

There is a specific way sets rot even when they are maintained. Every incident adds an item, so after two years the set is a museum of past bugs — and each of those items was chosen precisely because the system failed it, which makes the set systematically unrepresentative and systematically hard.

Hold a ratio. Something like: no more than a third of the set from incidents, the rest sampled from traffic on the refresh schedule. When incidents push past the ratio, either sample more traffic or move the older incident items into the regression suite, which is where they belonged all along.

The labels go stale too

A set can match today's traffic perfectly and still be wrong, because the answers have moved. The refund window changed, a product was discontinued, a policy was rewritten, a fee went up. Every reference answer touching those topics is now incorrect, and the evaluation is quietly penalising the right behaviour.

Two defences. Attach a policy version to the items that depend on one, as the reference-answers lesson described, and fail the run loudly when the versions diverge. And when a regression appears on a cluster of related items at once — five refund questions failing on the same day — check the policy before you check the model. Roughly half the time the model is right and the file is old.

Give it an owner and a date

Sets rot when nobody owns them. Put a name and a review date in the set's metadata file, review quarterly, and record what changed and why. It is fifteen minutes a quarter, and it is the difference between a measurement system and a folder of old JSON that everybody has stopped believing.

BEFORE YOU MOVE ON

A team replaces its evaluation set and the reported score drops from 87% to 81% with no change to the system. What should they publish?

A. The new 81%, since the new set better represents current traffic and the old figure is obsolete.
B. Both figures on the same system for one cycle, so the six-point gap is visible as a difference between sets rather than a change in quality.
C. The old 87% until enough history accumulates on the new set to make it comparable.
D. An average of the two, weighted by set size, to smooth the transition.

73 · 10 min

Fairness as arithmetic, and the choice you cannot avoid

THE ONE THING TO KEEP

Calibration and equalised odds cannot both hold when base rates differ, so fairness is a definition you choose and state, and a per-slice floor forces the work an average lets you avoid.

The overall number is an average over people

A system at 86% is not 86% for everybody. It is 91% for one group and 68% for another, and the average conceals which. That is not an ethical observation yet; it is a measurement failure, and it has an entirely concrete fix: compute the metric per group before you compute it overall.

What counts as a group depends on what you can legitimately observe. Often you can use non-sensitive attributes that carry most of the signal anyway:

- **language** and script — this is the big one for most products, and it is usually recorded
- **region or city tier**
- **device and connection quality** — long, garbled inputs from a cheap keyboard on a slow line
- **account age or tenure**
- **channel** — web, WhatsApp, phone transcript

Do not infer protected characteristics from names in order to measure them. Name-based inference is unreliable, and building the inference creates a dataset you then have to protect. Where legally collected demographic data exists, use it under the same rules as any other sensitive field. Where it does not, slice by what you have and say clearly which groups you could not check.

A finding worth expecting: a support assistant at 88% in English and 61% in Bengali is not one system with a quality problem. It is two systems, one of which was never evaluated.

One average, three systems

A floor of 80 fails this release; the average passes it. A floor cannot be met by improving the group that is already doing well, which is why it changes behaviour more than any statistical machinery.

Three definitions, and they conflict

When a system makes decisions about people — approving, flagging, prioritising, ranking — "fair" has to be made precise before it can be measured, and there is more than one way to do it.

Demographic parity. The positive rate is the same across groups. If 12% of group A is approved, 12% of group B is approved. Ignores whether the underlying rates genuinely differ.

Equalised odds. The error rates match across groups: the same true positive rate and the same false positive rate. Among people who should be approved, the same fraction are, in both groups.

Calibration within groups. A score of 0.8 means an 80% chance of the outcome, in every group. Everyone with the same score gets the same treatment.

All three sound like fairness. Here is the part that people find genuinely surprising the first time.

You cannot have all three

This is a mathematical result, not a matter of effort or engineering skill. Kleinberg, Mullainathan and Raghavan (2016) and Chouldechova (2016) showed independently that unless the base rates are identical across groups, or the classifier is perfect, calibration and equalised odds cannot both hold.

A sketch of why. Precision, recall and base rate are linked: `FPR = (1 − precision)/precision × (base rate)/(1 − base rate) × TPR`, roughly. Fix the base rates unequal and hold two of the fairness properties, and the third is forced to be violated. There is no clever model that escapes it.

The consequence is practical and it is the point of this lesson. **Fairness is a choice you state, not a box you tick.** You pick the definition that matches the harm you most want to avoid, you write down which one you chose and why, and you report the others so the trade-off is visible rather than hidden. A team that says "our system is fair" without naming a definition has not measured anything.

Which to choose, in practice: if a false positive is the harm — being wrongly flagged, wrongly refused, wrongly investigated — equalised odds is usually the definition you want, because it is about who bears the errors. If the score is handed to a human who will act on it as a probability, calibration matters more, because a score that means different things for different groups misleads every decision made from it.

Do it in practice

For a language-model feature, the whole exercise is often just this:

1. Tag every eval item with the slices you can observe.
2. Compute pass rate per slice, with an interval, and require at least 30 items per slice or say you cannot tell.
3. Publish the slice table in every report, worst slice first.
4. Set a floor: no slice below some stated level ships. A floor is a much better instrument than an average, because it cannot be met by improving the group that is already doing well.

That fourth point changes behaviour more than any statistical machinery. An average rewards work on the majority group; a floor forces work on the minority one.

What data can and cannot fix

Some gaps close with data: if a group is underrepresented in your retrieval corpus, few-shot examples or fine-tuning set, adding representative material helps and is the first thing to try.

Some gaps do not. If your labels themselves carry a historical bias — past decisions that were unfair, now encoded as ground truth — then a model that fits the labels perfectly reproduces the unfairness perfectly, and every metric will say it is working. This is the case where evaluation cannot save you, because your ruler is the problem. The only route through is to audit the labels themselves against a criterion that does not come from the same history, and that is a slow, human, contested piece of work.

Say so when it applies. "We measured X and cannot measure Y" is a much more useful sentence than a fairness dashboard implying you have covered it.

The free tools

Fairlearn (Microsoft, MIT) and **AI Fairness 360** (IBM, Apache 2.0) both compute the standard group metrics and disparity measures out of the box, and both are free. For a language-model feature, honestly, a `groupby` over

your per-item results gives you most of the value in five lines — which is another reason to store per-item records rather than summaries.

BEFORE YOU MOVE ON

A lending model is calibrated within both groups — a score of 0.7 means a 70% repayment rate for each — but has a higher false positive rate for group B. A reviewer says this proves the model is biased and calibration should be kept while equalising the error rates. What is the accurate response?

A. With unequal base rates, calibration and equalised odds cannot both hold, so the team must choose which to prioritise and state the choice and its cost.

B. The disparity is a data problem; adding more training examples from group B will bring both properties into line.

C. Calibration is the stronger property, so a calibrated model is fair by definition and the false positive gap is not a fairness issue.

D. The thresholds should be adjusted per group until the false positive rates match, which resolves the conflict.

74 · 8 min

Turning a complaint into a permanent test

THE ONE THING TO KEEP

Every incident becomes a minimised, provenanced test item, but keep those in a regression suite so the eval set still resembles real traffic rather than a museum of old bugs.

Every incident is a gap in the set

Something went wrong in production. A customer was told the wrong refund window; the assistant answered in the wrong language; a document nobody should see was retrieved and quoted.

The fix is the obvious part. The valuable part is the loop that makes sure that specific failure never returns unnoticed, and it is a five-step habit that most teams do not have.

The five steps

1. Capture the exact input. Not a description — the input, verbatim, with the retrieval context, the model version, the prompt version, and the timestamp. This is the reason the logging in the previous lesson exists. "A customer asked something about refunds" cannot be reproduced.

2. Reproduce it. Run the same input through the current system. Three outcomes, all informative:

- It reproduces. Good — you have a test.
- It does not, because the system is non-deterministic. Run it twenty times and record the rate. A failure that occurs one time in five is still a failure, and a one-shot test would have called it fixed.
- It does not, because something has already changed. Find out what, or your fix addresses a system that no longer exists.

3. Minimise. Strip the input to the smallest thing that still fails. A 40-turn conversation usually reduces to two turns. A minimal case is faster to run, easier to reason about, and much more likely to generalise to the family of inputs that share the cause.

4. Add it to the set, with provenance.

```
{"id": "inc-2026-08-14-refund-window",
 "input": "...",
 "expected": "states the 14-day window from the policy, or
asks",
 "assert": [{"type": "not_contains", "value": "30 days"}],
 "origin": "incident", "incident": "INC-2291",
 "added": "2026-08-14", "severity": "high"}
```

The `origin` field is what makes this manageable later. It lets you count how much of your set is history and how much is traffic.

5. Ask what would have caught it a week earlier. This is the question that produces the durable improvement, and the answer is always a mechanism, never a person. Sometimes it is a deterministic check. Sometimes it is a new rubric rule. Sometimes it is a monitor on an output metric nobody was watching. Write the answer down and build it, or the next incident of the same kind will also be found by a customer.

From a complaint to a permanent test

The origin field is what keeps the set from becoming a museum: it lets you count how much of it is history and how much is traffic, and hold incident items under a third.

Keep the set from becoming a museum

Here is the failure mode of doing this well. After eighteen months, 60% of your set is incident items — rare, hard, historical cases — and the set no longer looks anything like your traffic. Every number it produces is pessimistic, its slices are meaningless, and the team stops believing it.

Two controls:

- **Cap the proportion.** Keep incident-derived items under about a third of the main set. Move the overflow to a separate regression suite that runs in CI as a pass/fail gate and does not contribute to the quality percentage. Two files, two purposes: one estimates quality on real traffic, one prevents known bugs returning.
- **Refresh the base.** Every quarter, replace a slice of the ordinary items with a fresh sample of recent traffic. Keep the old ones in the frozen set for historical comparison, but let the live set track what people are actually asking.

The distinction is worth being firm about. A regression suite answers "has anything we already fixed come back?" and every item in it must pass. An eval

set answers "how good is this on the work we actually do?" and a percentage is the right output. Merging them gives you a number that is neither.

The review, and its tone

Hold a short review for anything customer-visible. Twenty minutes, four questions:

1. What happened, in one paragraph, with the timeline.
2. Why did the eval set not contain this case?
3. What check, monitor or rule would have caught it earlier?
4. What is now in the set, and who added it?

Question two is the one that improves the evaluation rather than the system, and it is usually the most interesting. The answer is often that the case existed in traffic and was never sampled, which points straight at your sampling, not at the model.

Keep it blameless, genuinely and not as a slogan. The moment an incident review becomes a search for who wrote the prompt, people stop reporting the small failures, and the small failures are your entire early-warning system.

BEFORE YOU MOVE ON

After eighteen months of diligently adding every incident to the eval set, a team finds their score has drifted down and no longer matches user satisfaction. What is the most likely explanation?

A. The judge has drifted over eighteen months and is now scoring more harshly than it did at the start.

B. The set is now dominated by rare historical failures, so it measures a distribution of hard cases rather than the traffic users actually send.

C. The system has genuinely got worse, and user satisfaction is a lagging indicator that has not caught up.

D. Incident items were added without expected outputs, so many of them are being scored as failures incorrectly.

75 · 8 min

Writing the result down

THE ONE THING TO KEEP

A result travels without you, so write one page carrying the set, the interval, the slices, the cost, what was not measured and the decision it supports.

The number leaves the room without you

You will say "about 86%, on a hundred items, plus or minus eight, and the Bengali slice is much worse". What travels to the next meeting is "86%". By the third retelling it is "nearly 90% accurate" and it is in a slide to a customer.

The only defence is a written artefact that goes wherever the number goes. One page. It takes fifteen minutes and it is the difference between a measurement and a rumour.

The result card

> ***Support reply quality — reply-v9***
>
> *2026-09-06 · owner: Aditi · dataset support-v4 (sha 9f21c0)*
>
> ***What was measured.*** *Whether a generated reply answers the question asked, makes no commitment absent from the policy text, and asks rather than assumes when information is missing. Six binary rules; two are gates.*
>
> ***Result.*** *86% of items passed all six rules (95% CI 79–91, n = 100, 3 runs, spread 2 points). Previous version, same items: 82% (74–89). 9 items improved, 4 regressed.*
>
> ***Gates.*** *0 policy-commitment violations in 300 runs. 1 account-fact violation, item tkt-0087, fixed and retained as a regression test.*

> ***By slice.*** *English 91% (n=61) · Hindi 84% (n=25) · Bengali 64% (n=14, interval too wide to act on — needs more items).*

> ***Cost and speed.*** *$0.0068 per reply, up from $0.0031. p95 latency 4.4s, up from 2.9s.*

> ***What this does not measure.*** *Whether customers were satisfied; whether the reply was appropriate in tone; anything about voice or attachments; anything in a language other than the three above.*

> ***Decision it supports.*** *Ship to the English and Hindi queues. Do not ship to Bengali until that slice has 40 items and clears 80%.*

Six headings. Every one of them exists because a team somewhere got burned by leaving it out.

The four sentences that make it honest

"On these hundred items." Never "the model is 86% accurate". Always "86% on this set". The set is the scope of the claim and it should be inseparable from it.

"Plus or minus eight." From module one. Any number without an interval invites comparison against another number without an interval, and that is how a team ships a three-point regression believing it is a three-point improvement.

"We did not measure X." The most valuable line on the page, and the hardest to write, because it reads as weakness and is the opposite. It stops somebody assuming you covered voice input, or Bengali, or tone.

"We cannot tell." For the Bengali slice with 14 items, the interval is roughly ±25 points. The honest report is not 64%. It is that the number could be anywhere from 40% to 88% and the only fix is more items. Reporting "we cannot tell yet" once buys more credibility than five confident numbers, because it demonstrates that the confident ones mean something.

Talking to people who will not read a confidence interval

Translate, do not simplify away:

- "About 8 in 10 replies are right, and we could be off by about 1 in 10 either way."
- "It's better than the old version on this set, by roughly 4 points, and 4 items got worse — we read those."
- "It works noticeably less well in Bengali, and we don't have enough Bengali examples yet to say how much less."

What not to do: report a mean judge score. "The assistant scores 8.4 out of 10" is meaningless outside your repository, it is not comparable to anything, it drifts when the judge model updates, and it will be quoted at you for a year. Judge scores are internal, relative, same-run comparisons. Keep them there.

Refusing to give a number

Sometimes the right answer to "what's the accuracy?" is "we don't have a measurement I would stand behind, and here is what it would take to get one — two days and a hundred labelled examples".

This is the hardest sentence in the course to say and the one that most defines whether the evaluation function in a company is trusted. A number produced under pressure without a set behind it is a guess wearing a decimal point, and it will be treated as fact by everybody who was not in the room.

Say the two days. In almost every case, somebody will find them.

BEFORE YOU MOVE ON

An executive asks for a single headline accuracy figure for a slide. Your set is 100 items, the pass rate is 86%, and one language slice of 14 items sits at 64%. What is the best response?

A. Give 86% and mention the language gap verbally, since a slide cannot carry caveats.

B. Give 82% instead, discounting for the weak slice, so the figure is conservative and defensible.

C. Decline to give any figure until the weak slice has enough items to be measured properly.

D. Give "86% on 100 support tickets, plus or minus 8", state that it covers English and Hindi only, and say the third language cannot yet be measured.

76 · 8 min

How much of this to actually do

THE ONE THING TO KEEP

Match the evaluation to the consequences of failure, and re-ask which tier you are in every time a prototype goes in front of customers.

Evaluation has a cost, and it is not small

Everything in this course is work. Roughly, at 2026 prices in time:

- A first 100-item labelled set, with a guide and a second labeller: **three to six hours**.
- A harness that runs, caches and diffs: **one to two days** for the first version.
- A calibrated judge, including the calibration set: **five hours**, then ten minutes a month.

- A human review programme: **eighty minutes a day**, indefinitely.
- Monitoring, dashboards and alerts: **two to three days**, then maintenance.
- Fairness slicing, red-teaming, agent environments: **days each**.

Do all of it for an internal tool used by five people and you have spent a fortnight measuring something that could have been replaced by asking those five people. Do none of it for a system that decides who gets a loan and you have built something nobody can defend.

The skill is proportionality, and it is a judgement about consequences, not about how interesting the model is.

Three tiers

Tier 1 — low stakes, reversible, small audience. An internal summariser, a drafting helper for your own team, a prototype.

- 20 to 30 examples, labelled by one person
- deterministic checks only: parses, complete, right language, no placeholder text
- a note in the repository saying what it is for and what it does badly

Half a day. Anything more is procedure for its own sake, and the users are close enough to tell you when it breaks.

Tier 2 — customer-facing, reversible. A support assistant, a search feature, a content generator with a human in the loop.

Everything in tier 1, plus:

- 100 items, two labellers, dev and test split
- a rubric and a calibrated judge, re-calibrated monthly
- the harness in CI: deterministic checks on every pull request, judge set nightly
- slice metrics with a floor, not just an average
- online proxies logged, with a weekly look

- a small review sample, random and targeted
- the incident loop

About two weeks to establish, then a few hours a week. This is where most work sits, and it is the tier this course is really written for.

Tier 3 — consequential or irreversible. Money moves, a medical or legal decision is influenced, content is published unreviewed, safety is at stake, or the volume is such that a 1% failure rate is thousands of people.

Everything in tier 2, plus:

- a much larger set, hundreds to thousands of items
- an adversarial suite with harmful-completion and false-refusal rates tracked together
- explicit fairness definitions, chosen and documented, with per-group floors
- continuous monitoring with alerts and an on-call owner
- sustained human review with double-labelling
- a staged rollout with a kill switch, and a written rollback threshold
- a named person accountable for the quality of the measurement itself

Months, and a permanent cost line. If that seems disproportionate, the tier assignment is the thing to argue about, not the list.

Choosing the tier honestly

Four questions settle it:

1. **Who is harmed when it fails, and how badly?** Not "is it embarrassing" — who, and how much.
2. **Can it be undone?** A wrong draft a human edits is recoverable. A sent message, a published post, an issued refund, a rejected application is not.
3. **Would anyone notice?** A failure that produces a complaint is self-limiting. A failure that produces a slightly wrong number in a report nobody checks can run for a year.

4. **How many people, how fast?** Volume converts a small rate into a large count.

The most common mistake is not under-evaluating a tier-3 system. It is running a tier-1 process on a tier-2 product because it started as a prototype and nobody re-asked the question when it went in front of customers. Re-ask it at every launch.

Where to start, if you start nowhere

In this order, and each step is useful before the next exists:

1. **Twenty labelled examples in a file.** One hour. It ends more arguments than anything else in the list.
2. **A script that runs them and prints a pass rate.** Two hours.
3. **Deterministic checks.** Two hours, and often the largest single quality gain in this list.
4. **Read fifty failures.** Ninety minutes, and it tells you what to do next.
5. **Grow to a hundred, split dev and test.** Then the rest of this course, in whatever order your failures demand.

The first three items are one day of work and they put you ahead of most teams shipping AI features. Not because the rest is unimportant, but because most teams have not done the first three, and are running on demos.

The thing to keep

The habit under all of it is small and portable: before believing a system works, say what would show that it does not, then go and look. Everything else in this course is machinery for doing that quickly, repeatedly, and in front of other people.

BEFORE YOU MOVE ON

An internal prototype with 30 labelled examples and a few deterministic checks has just been opened to customers as a beta. What is the most important change?

A. Re-assess the tier: customer exposure means a dev/test split, a calibrated judge, slice metrics with a floor, and an incident loop are now warranted.

B. Grow the set to 1,000 items so the pass rate has a tight confidence interval before more customers arrive.

C. Add an adversarial suite and continuous monitoring with an on-call owner, since the system is now public.

D. Keep the current setup and rely on beta customers reporting problems, since early users are tolerant and give direct feedback.

CASE STUDY · MODULE 8

Sixty days' notice and a set from January

A Pune fintech's support assistant when the provider deprecates the pinned model, complaints are rising, and the eval set is eight months old

PaisaSathi runs a lending app with about two million users and a support assistant that answers questions about EMI dates, statements, KYC status and repayment. The assistant had been stable for most of a year. The team had done the early work: a frozen eval set of 300 items sampled in January, a calibrated judge, a canary set of twenty items run hourly, and a pinned model version in a config file.

In the first week of September two things arrived within days of each other. The model provider sent a deprecation notice: the pinned version would be withdrawn in sixty days, and the recommended replacement was a newer model under a rolling alias. And the head of support, Priya, reported that complaints tagged "assistant unhelpful" had risen by 40% over six weeks, with nothing deployed on the assistant in that time.

The engineering lead, Rohan, had three candidate explanations and no evidence for any of them. The provider might have changed something under the pinned version despite the pin. The users might have changed. Or nothing had changed and the complaint tag was being used differently.

The decision that could not wait was what to do about the deprecation. The product manager wanted to switch to the new alias immediately: the provider's announcement described it as better on every benchmark, the deprecation was going to force the move anyway, and moving now might fix the complaints. Rohan wanted to hold the pinned version, run the frozen set against both models, read the per-item diff by slice, re-tune the prompt for the new model, and then canary. He estimated four working days. The cost of his four days was four more days of rising complaints if the model was the cause. The cost of switching immediately was that whatever the new model did differently, nobody would know until users did.

The canary set answered the first question in an hour. Its twenty items had produced byte-identical outputs for eleven weeks. The pinned model had not changed. Whatever was driving the complaints, it was not the provider.

So Rohan looked at the inputs. He bucketed August's traffic and the January eval set on language, intent and input length, and computed the total variation distance between the two mixes. It was 0.27. In January, Marathi-language messages had been 4% of traffic; in August they were 19%, following a marketing campaign in Maharashtra's tier-two cities. The eval set contained twelve Marathi items. The assistant's pass rate on those twelve was 58%, which nobody had looked at because twelve items in a slice of 300 barely moved the headline.

That reframed both problems. The complaints were most likely a mix shift onto a slice the assistant handled badly and the set could not see. And the deprecation decision now had a specific question attached: was the new model better or worse in Marathi, and at what latency and cost?

The frozen set was the only asset that could answer it, and it was stale. Refreshing it would break comparability with every number the team had reported since January. Not refreshing it meant deciding a model migration on a set that no longer resembled the users.

What order would you do things in, and what would you report to Priya?

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

Rohan did not refresh the frozen set. He drew a second set, 200 items from August's traffic, stratified to the current mix, labelled over two days by two support agents. Both sets were run against both models, and the report carried all four numbers.

On the January set, the new model scored 84% against the old model's 81%. On the August set, 79% against 76%. The three-point gap held on both, which was reassuring, but the slice table was the finding. In Marathi the new model scored 71% against 58%. Its refusal rate on borderline-but-legitimate Hinglish messages, the kind that mention loans and stress in the same sentence, was 9% against 2%. And its p95 latency was 5.1 seconds against 2.3.

The prompt was re-tuned for the new model over two days, mainly by removing formatting instructions the old model had needed, and the refusal rate on the Hinglish slice came down to 4%. A per-request token cap and a five-second timeout, counted as failures in every quality metric, brought p95 to 3.4 seconds. Then a 5% canary for two business days, with refusal rate and escalation rate as guardrails, and a full ramp over the following week.

The set policy was written down for the first time. The January set stayed frozen as the historical yardstick. The August set became the headline set, versioned and dated, with both reported for one cycle so the six-point difference between them was published as a fact about the sets rather than mistaken for a regression. Incident-derived items were capped at a third. The monthly divergence check became a scheduled job, and Marathi got its own floor.

Priya's report said three things: the model had not changed; the users had, and the assistant had been weak for the new users all along; and the migration improved the weakest slice by thirteen points, with the interval stated.

The canary set had produced identical outputs for eleven weeks. What did that establish, and what would the team have concluded without it?

Twenty fixed items run hourly with stored outputs showed the pinned model had not changed, which eliminated the provider as the cause of rising complaints in an hour. Without it, the team would most likely have blamed the provider, switched models under pressure, and never looked at the inputs, where the real change was.

Rohan drew a new set rather than refreshing the frozen one, and reported both. Why does a silent refresh do damage, and what did reporting both buy?

Changing a set changes the number, and a number that moved because the set moved looks exactly like a quality change. Keeping January frozen preserved every historical comparison, and running both models on both sets let the three-point migration gain be separated from the six-point difference between the sets. Publishing the gap between sets turned it from a phantom regression into a stated fact.

The migration decision was made on a slice table rather than on the headline. What in the table would have been invisible in a single number, and how did it change what shipped?

The headline said the new model was three points better. The slices showed a thirteen-point gain in Marathi, a refusal rate that had more than quadrupled on a sensitive Hinglish slice, and a p95 latency that had doubled. Each drove a different action: the Marathi gain justified the move, the refusal rate was fixed by re-tuning the prompt, and the latency was bounded with a token cap and a timeout counted as failure, before any user saw the new model.

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 Why does the course insist on reporting p95 latency rather than the mean?

A. The mean hides the one user in twenty who waits nine seconds.

B. The mean cannot be computed for a right-skewed distribution such as generation time.

C. p95 is the figure the provider bills against, so it is the one that maps to cost.

D. The mean is always higher than p95 for a generative system, so it overstates the problem.

2 Refusal rate moves from 2% to 9% overnight with no deployment on your side. What is the most likely cause?

A. Users began sending markedly more abusive messages that night.

B. A provider-side change to the model behind an alias.

C. The judge model drifted and is now scoring more outputs as refusals.

D. The offline evaluation set has gone stale.

3 Generated summaries start being saved into the help centre that the retriever indexes. What happens to the groundedness metric?

A. It falls, because machine-written text is less internally consistent than human text.

B. It becomes impossible to compute, since claims can no longer be matched to passages.

C. Nothing changes, because groundedness is a property of the answer and not of the corpus it was drawn from.

D. It stays high while truth drifts, since answers are grounded in machine-written material.

4 A refreshed evaluation set scores 81% on a system that scored 87% on the old set. What does the course say to do?

A. Report both versions for one cycle and publish the gap as a fact about the sets.

B. Treat it as a six-point regression and roll the system back to the previous release.

C. Discard the new set as mis-labelled, since the system did not change.

D. Average the two figures and report 84% as the current quality.

5 Two groups have different base rates for the outcome a classifier predicts. Which statement about fairness definitions is correct?

A. Calibration within groups and equalised odds can both hold, given enough training data and a flexible enough model.

B. Demographic parity guarantees both calibration and equalised odds.

C. They cannot both hold unless the classifier is perfect; choose one, state it, report the others.

D. Fairness is achieved by removing the group column from the features.

6 After two years, 60% of an evaluation set is made of items added from incidents. What is the problem, and the fix?

A. No problem: incident items are the most valuable items in any set, because each one was a real failure seen by a real user.

B. The set no longer resembles traffic; cap incident items near a third and move the overflow to a regression suite.

C. Delete every incident item older than a year so the set stays current.

D. Reweight the incident items down to their share of live traffic before computing the score.

EXAMINATION

Knowing If It Works: end-of-course examination

This paper covers the whole course: building a labelled set and reading its uncertainty, choosing a metric that matches the decision, deciding whether a difference is real, evaluating the systems you are actually handed, automating judgement without being misled by it, evaluation inside the working loop, evidence other people can act on, and keeping a shipped system honest over time. Each attempt draws 25 questions from a larger pool, with every module represented. The pass mark is 70 per cent. There is no timer; take as long as you need. If you do not pass, you can retake after 30 minutes with a fresh draw of questions. Passing opens the next course in the track, and a teacher can open it for you regardless of the result.

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 counts as evidence`

An intent router scores 78%. Always predicting the most common class scores 61%, twenty keyword rules score 74%, and two humans agree 88% of the time. What is the language model's contribution over the cheap baseline?

A. Seventeen points, the gap above the trivial baseline of always predicting billing.

B. Ten points, the distance from the model up to the ceiling where the two humans agree.

C. Four points, over the twenty keyword rules.

D. Seventy-eight points, since the baselines are context and not part of the result.

2 1. What counts as evidence

A team's offline pass rate rose eight points after a prompt change, and the monthly tickets-resolved figure it was supposed to predict did not move. What has been learned?

A. The leading metric's claim to predict the real one has failed a check.

B. One month is too short for a lagging metric to move, and the team should wait another quarter before reading it.

C. The lagging metric is too noisy to be useful and the offline number should be trusted instead.

D. The evaluation set needs more items before eight points can be believed.

3 1. What counts as evidence

After normalising and hashing, one string accounts for 6% of an evaluation set. What does the course say that means?

A. The string is a common case and should be kept in proportion so the headline reflects real traffic.

B. The set is fine, since 6% is below the level at which duplicates begin to distort a pass rate.

C. The string should be moved to the test set so it cannot inflate the development score.

D. It is not a set but a template; keep one copy and record its frequency.

4 1. What counts as evidence

A labelling guide has a one-sentence definition, three passing examples and three failing examples. Which element does the course say is most often skipped and does the most work?

A. A numeric scale from one to five so that partial quality can be recorded rather than forced into pass or fail.

B. Two boundary cases, each with a written ruling.

C. A list of the labellers' names and qualifications for the record.

D. A description of the model being evaluated so labellers know what to expect.

5 1. What counts as evidence

A system's score has plateaued at about 90% on a hand-labelled set. What does the course say the next honest step is?

A. Try another prompt variant, since a plateau shows the current wording has been exhausted.
B. Switch to a larger model, because a plateau at 90% is usually a capability ceiling rather than a data problem.
C. Re-label the failed items blind and count how many labels were wrong.
D. Add three hundred more items so the last ten points can be measured precisely.

6 1. What counts as evidence

Old prompt 82 of 100, new prompt 88 of 100 on the same items. Two items regressed and eight improved. What does the paired rule of thumb say?

A. Not quite conclusive: 6 falls just short of $2\sqrt{10}$, about 6.3.
B. The six-point gap is real, because it exceeds the two-point difference between the flip counts.
C. The gap is noise, because six points sits inside the plus or minus eight interval on a hundred items.
D. The comparison cannot be made, because fewer than twenty items changed verdict.

7 1. What counts as evidence

A model will run on next month's data. Compared with a random split, a time-based split, with the last four weeks as the test set, usually gives a lower score. Why is the lower number the honest one?

A. Because the last four weeks contain fewer items, so the estimate is more conservative.
B. Because recent data is always harder than older data, so any set built from it scores lower than a random one.
C. Because a time split removes near-duplicates automatically, which a random split does not.
D. Because inputs drift, and a random split tests on the same period the system was tuned on.

8 2. Metrics that mean something

A ticket router has a macro-F1 of 0.52 and a micro-F1 of 0.88 at the same time. What does that pair of numbers tell you?

A. One of the two figures has been computed wrongly, since the two averages should agree with each other on a single-label task.

B. Common categories are handled well and rare ones badly; only macro shows the rare queue is broken.

C. The router is over-fitted, because macro averages are always lower on held-out data.

D. The classes are balanced, which is the only case in which the two averages diverge.

9 2. Metrics that mean something

Items carry five labels each, and every label is predicted correctly 90% of the time, independently. Exact-match scoring reports 59%. What should the team conclude?

A. The system is badly broken on about two items in five and needs retraining before anything else is attempted on it.

B. The per-label figure is wrong, since a 90% system cannot score 59% on any honest metric.

C. The labels are not independent, and the exact-match figure should be raised to match the per-label one.

D. Nothing is broken; the metric multiplies, and exact match belongs only where the whole set must be right.

10 2. Metrics that mean something

Three positives score 0.9, 0.6 and 0.4. Four negatives score 0.8, 0.5, 0.3 and 0.1. What is the ROC-AUC?

A. 0.75: nine of twelve positive-negative pairs rank the positive higher.

B. 0.86, the mean of the three positives' scores, since AUC summarises how high positives score on average.

C. 0.67, because two of the three positives outrank the highest-scoring negative.

D. It cannot be computed without plotting the full curve at every threshold.

11 2. Metrics that mean something

With 60 harmful posts in 1,000, lowering a threshold takes false positives from 32 to 153. The ROC curve barely moves. Why, and what should be reported instead?

A. Because the ROC curve is plotted on a logarithmic scale; report AUC to more decimal places so that the change becomes visible.

B. Because the ROC curve reflects only recall; report accuracy at the new threshold instead.

C. The false-positive rate divides by 940 negatives, so it barely moves; report the PR curve and average precision.

D. Because thresholds below 0.5 fall outside the region the curve covers; report F1 at the default.

12 2. Metrics that mean something

A language-model classifier is asked to state its confidence from 0 to 100 and answers 92 for almost everything, right or wrong. What should be done if a usable score is needed?

A. Apply Platt scaling to the stated confidence, which will spread the clustered values out across the full range.

B. Take token log-probabilities of the label tokens where exposed, and calibrate those.

C. Average the stated confidence over three samples to reduce the noise in the figure.

D. Lower the temperature so that the model becomes more honest about its own uncertainty.

13 2. Metrics that mean something

A demand forecast has a MASE of 1.0. What has the team built?

A. A forecast that is perfectly calibrated, since the ratio is exactly one.

B. A forecast whose errors sum to zero over the period.

C. A model that is 100% better than always predicting the mean.

D. A machine that repeats last week.

14 2. Metrics that mean something

System A answers 88% of answerable questions correctly and abstains on 15% of unanswerable ones. System B answers 79% correctly and abstains on 90%. Which does the course call more dangerous, and why?

A. System A: it invents fluent answers where the corpus is silent, with a citation attached.

B. System B, because a 79% accuracy loses the user's trust faster than an occasional invented answer ever would.

C. Neither; the two numbers measure different things and cannot be set against each other.

D. System B, because refusing 90% of anything is a sign the abstention threshold is broken.

15 3. Deciding whether a difference is real

Before trusting a comparison harness, the course says to run configuration A against a second run of configuration A. The harness reports a significant difference. What has been found?

A. A real effect of non-determinism, which should be reported as the floor below which no improvement can be claimed.

B. Evidence that the significance level is too loose and should be tightened to 0.01.

C. Nothing unusual; A against A produces a significant result about half the time.

D. A bug in the harness: dropped items, misaligned rows, a cache, or a truncated ordered set.

16 3. Deciding whether a difference is real

A comparison reports p = 0.03. Which statement is correct?

A. There is a 3% chance that the two systems are in fact equally good, and a 97% chance that they differ.

B. If the systems were equally good, a gap this large would appear about 3% of the time.

C. There is a 97% chance that the new system is the better one.

D. The improvement is large enough to be worth shipping.

17 3. Deciding whether a difference is real

A comparison gives p = 0.31 and an interval on the difference of −3 to +7 points. Which is the honest statement?

A. The two systems are the same, since the result is not significant at the conventional level.

B. The new system is probably 2 points better, since that is the midpoint of the interval and therefore the best estimate.

C. The experiment could not distinguish them; it rules out a regression above 3 and a gain above 7.

D. The comparison should be re-run until the p-value drops below 0.05.

18 3. Deciding whether a difference is real

An evaluation has 500 conversations from 40 customers, about 12.5 per customer, with an intra-cluster correlation of 0.3. Roughly what is the effective sample size?

A. About 110, since the design effect is about 4.45.

B. About 500, since each conversation is a separately logged observation with its own outcome.

C. About 250, halved to allow for the clustering.

D. Exactly 40, the number of customers.

19 3. Deciding whether a difference is real

Twenty slices are tested with Benjamini-Hochberg at q = 0.10. The smallest p-value is 0.004 and the fifth smallest is 0.03. Which of the two is accepted?

A. Both, since each is below 0.05 and Benjamini-Hochberg only reorders the tests rather than changing the level.

B. Neither, since both exceed the Bonferroni level of 0.0025.

C. Only the 0.03, since the procedure works from the largest rank downwards.

D. Only the 0.004: rank 1 is tested against 0.005, and rank 5 against 0.025.

20 3. Deciding whether a difference is real

A per-item covariate, the score of a cheap baseline on the same item, correlates at 0.7 with the outcome and is unaffected by the change. What does adjusting for it do to the variance of the estimate?

A. Reduces it by about 30%, in proportion to the correlation between the covariate and the outcome.

B. Reduces it by about 51%, since the factor is $1 - 0.7^2$; like doubling the items.

C. Reduces it by about 70%, since the covariate explains that share of the outcome's variation.

D. Changes nothing, because a covariate cannot alter the expectation of the metric being estimated.

21 3. Deciding whether a difference is real

A team has a budget of 1,000 model calls and wants to know which of two systems is better overall on a non-deterministic task. How should the budget be spent?

A. 200 items run five times each, so that the run-to-run noise is averaged out before the two systems are compared.

B. 100 items run ten times each, to get a reliable per-item verdict on every item.

C. 1,000 items once; for an average, one run on many items beats repeats on few.

D. 500 items twice, as a compromise between item variance and run variance.

22 4. The systems you will actually be handed

For a Hindi or Tamil translation system, why does the course make chrF the default over BLEU?

A. It scores character n-grams, so a correct but differently inflected translation is not punished.

B. It was trained on Indian language pairs and so correlates better with human judgement there than BLEU does.

C. It uses several references at once, while BLEU is limited to one.

D. It is computed per sentence, whereas BLEU is only meaningful over a corpus.

23 4. The systems you will actually be handed

A simulated user gives a multi-turn assistant an 88% task-success rate. Twenty real transcripts scored on the same session rubric give 71%. What is the simulator good for?

A. Nothing; a seventeen-point gap shows the simulator is broken and it should be discarded entirely.

B. Headline numbers, once 17 points are subtracted as a fixed correction factor.

C. A regression detector for comparing versions; the headline comes from real transcripts.

D. Replacing real transcripts, since it produces hundreds of sessions cheaply.

24 4. The systems you will actually be handed

In an extraction evaluation, exact string match gives 81% field accuracy and normalised match gives 93%. What does the twelve-point gap measure?

A. The model's tendency to invent values that look like the gold ones but are not in the document.

B. Formatting debt: dates and amounts written differently, not read wrongly.

C. Label noise in the gold file, which normalisation happens to paper over.

D. The share of fields the model left blank.

25 4. The systems you will actually be handed

Why can interleaving two rankers settle a comparison in a day when an A/B test needs a fortnight?

A. Interleaving doubles the number of clicks collected per user, so the sample fills twice as fast.

B. Interleaving removes position bias, which is what slows an A/B test down.

C. Interleaving measures absolute relevance rather than a preference between rankings.

D. Every user sees both rankers, so between-user variance drops out.

26 4. The systems you will actually be handed

Two teams report word error rates of 8% and 14% on identical audio and identical transcripts. What is the most likely cause?

A. Different text normalisation: casing, numerals, fillers, contractions, transliteration.

B. One team's recordings were down-sampled to telephone bandwidth before scoring, which roughly doubles the error.

C. One team used character error rate and mislabelled it as word error rate.

D. Run-to-run variance in the speech model, which is larger than for text models.

27 4. The systems you will actually be handed

A document pipeline is live and there are no labels for production traffic. Which check can still measure reading errors on every document?

A. Character error rate against the OCR engine's own confidence score, which needs no transcript.

B. A judge model asked whether the extracted record looks plausible for that document type.

C. Internal consistency: line items sum to the total, tax matches the rate, dates are ordered.

D. Comparing each document's fields against the previous document received from the same supplier.

28 4. The systems you will actually be handed

A learning curve for a tabular model flattens at 40% of the available training rows. What does that say about collecting more data?

A. More rows will help, since the model has only used 40% of what exists and the rest is already collected.

B. More rows will not help; look to features, the target definition, or accept the ceiling.

C. The remaining 60% of rows are mislabelled and should be audited before use.

D. The model has over-fitted and needs stronger regularisation.

29 5. Automating judgement

A prompt change raises a judge's average score. What is the cheapest check that the improvement is quality rather than verbosity?

A. Re-run the judge with a different random seed and see whether the gain persists across seeds.

B. Ask the judge to rate the same outputs on a ten-point scale instead of five.

C. Plot judge score against output length and look at the correlation.

D. Compare the new outputs against the old ones with an embedding similarity score.

30 5. Automating judgement

A rubric has six binary rules, two of which are gates. Why does the course refuse to report a single average across the six?

A. An average lets a gate fail 5% of the time and still report 0.95.

B. Binary rules cannot be averaged in any meaningful way; only scales can be combined into a mean.

C. The judge scores gates on a different scale from the other rules.

D. Six rules is too few for an average to be stable across runs.

31 5. Automating judgement

A rubric rule has never failed on any of 500 items. What does the course say about it?

A. It is a well-designed rule and should be kept as a safety net against a failure that has not yet appeared.

B. The judge is too lenient on it and the prompt should be tightened until it fails sometimes.

C. The 500 items are too easy and should be replaced with harder ones.

D. Retire it, or move it to a cheap deterministic check.

32 5. Automating judgement

Violations are 4% of traffic. A team builds a judge calibration set of 50 random items. What is wrong with that?

A. Nothing; 50 items is the size the course recommends for a calibration set, and a random draw is the honest one.

B. It holds about two violations; enrich it with 25 known violations and 25 clean items.

C. It should be 500 items, since a judge needs ten times more calibration data than a human labeller.

D. It should be drawn from synthetic items so that violations can be generated on demand.

33 5. Automating judgement

A retrieval assistant's groundedness is 100% and users are being told a refund window that changed six months ago. What has happened, and what finds it?

A. The judge has drifted since it was calibrated; re-calibrate it against fresh labels before anything else.

B. Retrieval is failing and the model is inventing; measure recall at k.

C. It is faithfully citing a superseded document; only a check on the corpus finds it.

D. The claim decomposition is too coarse; split answers into finer claims.

34 5. Automating judgement

A team re-runs its judge harness after editing documentation, and the bill is as large as a full run. Which cache key would have made the re-run free?

A. Item id, output text, rubric version and judge model.

B. The item id and the run's timestamp, so that each run is cached separately.

C. The git commit, so that any code change invalidates everything cached.

D. The prompt template alone.

35 5. Automating judgement

For a system with retrieval and tools, which prompt-injection test does the course say is most often left out?

A. Direct requests in the user message framed as roleplay, fiction or a research paper.
B. Requests that ask the model to speak as a named real person.
C. A user in evident distress whose message looks like a policy violation.
D. Instructions hidden inside documents the system reads.

36 6. Evaluation in the working loop

During a harness run some items fail with a 429 rate-limit error. How should they be handled?

A. Score them zero, since the system did not produce an answer and the user would have seen nothing.
B. Retry with backoff and count them separately; a rate limit is not a quality failure.
C. Drop them silently so the pass rate reflects only completed items.
D. Retry them until the answer passes the checks.

37 6. Evaluation in the working loop

How should the failure band for the blocking core set in CI be set?

A. At a round number the team agrees on, such as must be above 85%, so that everyone knows where the line is.
B. At zero tolerance: any drop against the previous run blocks the merge until somebody explains it.
C. At the width of the bootstrap interval on the full nightly set.
D. From the run-to-run spread measured once with an A/A test, written into the config with the date.

38 6. Evaluation in the working loop

Which habit does the regression-testing lesson call the highest-value one, and what does it catch that metrics miss?

A. Committing generated outputs for a few items, so a prompt change is a readable diff.

B. Running the full set on every commit, which catches regressions before they can be merged.

C. Setting a hard threshold at three points, which catches every meaningful drop without producing false alarms.

D. Logging the aggregate score over time, which catches slow decline.

39 6. Evaluation in the working loop

A team wants to put regeneration rate on a quality dashboard. What does the course say has to happen first?

A. Nothing; regeneration is unambiguous and can be watched hourly straight away.

B. Replace it with the thumbs-up rate, which is an explicit signal from the user.

C. Check it against human quality labels on about 100 sessions first.

D. Wait a quarter for enough data to establish a stable baseline before any of it is shown.

40 6. Evaluation in the working loop

A review programme reads only items the judge flagged, items with low retrieval scores and items users complained about. What does the course say it will miss?

A. Nothing important, since flagged, low-scoring and complained-about items are exactly where the problems are.

B. The failure modes nobody has thought of, which only a random sample finds.

C. The most common failures, which are under-represented among flagged items.

D. Failures in the long tail of rare queries.

41 6. Evaluation in the working loop

A product has 400 sessions a week and wants to know whether a rewrite improves resolution by three points. What does the course call the least honest option?

A. Interleaving both variants for the same users.

B. Six weeks before and six weeks after, with eyes open about what else changed in the meantime.

C. Ten conversations with real users about the new replies.

D. Running the underpowered A/B test and reading its result.

42 6. Evaluation in the working loop

A guardrail metric fires on about one request in a thousand. A 1% canary on 50,000 daily requests is proposed. Why is that too small?

A. It yields about 500 requests and one expected event a day, too slow to trigger anything.

B. Because 1% of users is not enough for sticky assignment to work across a session.

C. Because a canary must always be at least 5% to be representative across regions and platforms.

D. Because the guardrail should be measured in shadow mode rather than on a live canary.

43 7. Evidence other people can act on

An internal system card carries supplier terms and unfixed vulnerabilities that cannot be published. What rule governs the external version?

A. It must be identical to the internal card, or it is not a card at all.

B. It may round the numbers upwards to avoid alarming customers.

C. It should carry the headline accuracy only, with detail available on request.

D. It may omit, and may never contradict.

44 7. Evidence other people can act on

Why does the vendor lesson recommend re-running your evaluation set on a second provider every quarter?

A. To keep the second provider's free tier active and the account in good standing for use in an emergency.
B. A tested fallback is an operational option, and it shows whether prompts have become provider-specific.
C. Because most providers require it as part of their terms of service for enterprise accounts.
D. To compare benchmark ranks between the two providers on the same date.

45 7. Evidence other people can act on

A team's own score is 88%. A held-out set owned by a different person, whose items the builders never see, gives 74%. What does the course call the gap?

A. A sign the held-out set was sampled from harder traffic and should be rebalanced to match.
B. Run-to-run variance, which is larger than teams expect on held-out data.
C. The size of the team's self-deception.
D. Evidence that the held-out owner labelled to a stricter standard.

46 7. Evidence other people can act on

A private evaluation set cannot be verified by anyone; a public one decays into the training data within a year. What compromise does the course propose?

A. Publish the method and a sample; hold the rest; rotate a portion yearly.
B. Publish the whole set and refresh it every month so that contamination never has time to matter.
C. Keep it entirely private and report only the methodology.
D. Publish the set only to a certified auditor under a non-disclosure agreement.

47 7. Evidence other people can act on

Reading across the regulatory frameworks, which practical consequence does the course say surprises teams?

A. That accuracy above 95% is required for any high-risk use before it may be deployed.
B. That every model must be certified by an external auditor before launch.
C. That user content may never, under any legal basis, be used to build an evaluation set or a calibration set.
D. Old versions' evaluation artefacts must be kept, to answer how you knew it worked.

48 7. Evidence other people can act on

After a prompt change, the number of cases where a guardrail caught something before it reached a user doubles, and no harm reached anybody. How should that be read?

A. As reassurance: the guardrail is working exactly as designed and catching what it should.
B. As a finding: near misses rise before incidents do.
C. As noise, since near misses are too rare to be a metric.
D. As a reason to loosen the guardrail, since it is firing too often.

49 7. Evidence other people can act on

A support assistant is 90% correct on its evaluation set and resolves 41% of problems in production. The funnel shows the largest loss between response was correct and user acted on it. What does that argue for?

A. A better model, since the correct-answer rate is the root of the funnel and everything below it inherits its errors.
B. A larger evaluation set, since the offline number is evidently unreliable.
C. Interface and writing changes: show the source, be shorter, give a next step.
D. A satisfaction survey at the end of every conversation.

50 7. Evidence other people can act on

Five annotators label toxicity; the majority vote becomes the gold label. What does the course say is lost, and what practice preserves it?

A. The minority reading; keep every annotator's individual labels.
B. Nothing is lost; majority vote is the standard and accepted way to resolve disagreement.
C. Statistical power, which is restored by adding more annotators.
D. Speed; a single expert annotator is faster and just as accurate.

51 8. Living with it

A proposed model scores three points higher than the current one and its p95 latency goes from 2.4 to 8.9 seconds. A smaller model scores four points lower at six times lower cost. How should the row be read?

A. Three points is usually a bad trade for a person waiting; the cheap row deserves a look once cost is a monthly bill.
B. Ship the higher-scoring model, because three points of quality outweighs any amount of latency over the long run.
C. Neither alternative, since quality, latency and cost cannot be compared on one table.
D. Ship the cheap model, because cost always dominates once a product has scale.

52 8. Living with it

Requests that time out at five seconds are dropped from the quality metric. Why does that bias the number?

A. It does not; timeouts are infrastructure events and belong on a separate dashboard from quality.
B. Dropping them lowers the sample size, which widens the interval but does not shift it.
C. Slow requests are disproportionately the hard ones, so the metric describes the survivors.
D. Timeouts are random, so dropping them only adds noise and never a bias.

53 8. Living with it

A monitoring alert fires every Monday morning and the team has started ignoring it. Which rule from the monitoring lesson is being broken?

A. Alert on the mean rather than on percentiles, since the mean is more stable across the week.

B. Compare with the same hour last week, with a minimum denominator and a sustained change.

C. Run judge calls on 100% of traffic rather than a sample.

D. Route alerts to the whole team rather than to a single owner.

54 8. Living with it

A team retrains on suggestions users accepted, treating acceptance as a correct label. What does the course say that teaches the model?

A. The user's preferences, which is exactly what fine-tuning on feedback is supposed to capture.

B. Nothing new, since accepted outputs were already in the training distribution.

C. To be shorter, because users accept short suggestions more often.

D. Its own plausible errors, since users accept the mistakes they could not detect.

55 8. Living with it

Bucketing the evaluation set and last month's traffic by intent, language and length gives a total variation distance of 0.27. What does the course say that means?

A. The headline is drifting from reality and a refresh is due before it reaches 0.3.

B. The set still resembles traffic; values under 0.3 are acceptable and no action is needed this month.

C. The traffic has changed too much for any evaluation set to be built from it.

D. The number is meaningless without at least ten buckets.

56 8. Living with it

Why does the fairness lesson say a per-slice floor changes behaviour more than any statistical machinery?

A. Because a floor is easier to compute than a confidence interval and easier to explain to a product manager.

B. Because regulators require floors rather than averages.

C. A floor cannot be met by improving the group already doing well, so it forces work on the worst slice.

D. Because averages are biased upwards by large slices.

57 8. Living with it

A Bengali slice has 14 items and a pass rate of 64%. How should the result card report it?

A. As 64%, since that is what was measured and a figure without an interval is better than none at all.

B. As a range of roughly 40% to 88%, with the statement that the team cannot tell yet.

C. As an average with the Hindi slice, to reach a usable sample size.

D. Omitted, since a slice under 30 items should not appear on a card.

58 8. Living with it

According to the final lesson, what is the most common proportionality mistake?

A. Under-evaluating a system that moves money or influences a medical decision, because the team is small.

B. Over-evaluating an internal tool used by five people.

C. Choosing tier three for everything to be safe.

D. Keeping a prototype's process on a product after it went in front of customers.

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. A demo is not a measurement — A

`B` — Reads a tie on examples neither version fails as evidence that the two versions are equivalent.

`C` — Treats absence of a measured gain as evidence the change is useless, when nothing was capable of measuring a gain.

`D` — Assumes random sampling is the missing ingredient, when the deeper issue is that the set has no failures left to move.

2. The decision the number has to serve — C

`A` — Confuses breadth of measurement with a stated commitment; eight metrics after the fact give eight ways to find a favourable one.

`B` — Records the configuration, which matters, but nothing about it constrains how the result will be interpreted.

`D` — Adds reviewer effort without fixing the underlying problem: two people with no stated criterion still disagree about what they are looking for.

3. A hundred examples, made by hand — A

`B` — Applies the rule against tuning a set to your results so literally that known-wrong labels are preserved as truth.

`C` — Mistakes hard-but-real cases for noise, and makes the set quietly easier every time it happens.

`D` — Patches the score instead of the data, so the next person to run the set inherits the same eight wrong labels.

4. Where the examples come from — B

A — Confuses coverage with representativeness: stratifying buys resolution on rare strata, and by construction destroys the traffic proportions the average depends on.

C — Invents a restriction; stratified samples support any metric, provided the strata are reported separately or weighted back.

D — Treats a bias as a sample-size problem; ten times the items sampled in the same proportions gives the same distorted average, just with tighter error bars around the wrong figure.

5. Writing a labelling guide two people can agree on — B

A — Reads raw agreement at face value when almost all items are negatives, so most of that 95.5% is agreement about the easy cases.

C — Treats disagreements as an inconvenience to be settled by authority rather than as the signal that shows where the guide is silent.

D — Reaches for more data when the immediate problem is which statistic is being read, not how much of it there is.

6. How much your labellers actually agree — C

A — Reads a low kappa as a verdict on the people, when the arithmetic here is dominated by the class imbalance rather than by reviewer behaviour.

B — Fixes the problem by dropping the inconvenient statistic; 96% agreement is exactly what two people who almost always say "no" would get by accident.

D — Assumes half the disagreements are noise on no evidence; with only two true positives expected in 100 items, four disagreements is a large signal about the definition, not a small one.

7. Test cases you generate, and what they cannot tell you — B

A — Treats a distribution mismatch as sampling error; more items from the same generator move the number nowhere, because the generator's distribution is what differs from traffic.

C — Possible in practice but a different failure; even with perfect labels the generated set would still misstate the traffic mix, which is what produced this 17-point gap.

D — Names one plausible slice as the whole cause; the dominant effect is the easier distribution overall, and abuse is usually a small fraction of live volume.

8. Two sets, and the ways they get contaminated — D

A — Blames sample size when 4,000 items give an interval of roughly a point, far too tight to explain a 23-point gap.

B — Names a real hazard, but repeated test runs would push dev and test apart, not push both above live performance together.

C — Accepts a distribution-shift story without checking the structural flaw sitting in plain sight in the way the split was made.

9. The number above it and the number below it — B

A — The interval matters, but it does not change the interpretation here: even the top of the interval is beaten by a system that never flags anything.

C — An improvement on accuracy, but it is still a score with nothing to compare it against; without the trivial baseline you cannot say whether any of the model's behaviour is doing work.

D — The ceiling matters for interpreting near-ceiling results; this result is not near the ceiling, it is barely above the floor, and the floor is the figure that reframes it.

10. How wrong your number probably is — A

B — Applies the interval for a single independent rate to a paired comparison, discarding exactly the information that makes the comparison sensitive.

C — Nets the flips into a single count and then judges it by intuition, when the test is about the imbalance between the two flip directions.

D — Treats sample size as the only route to confidence and misses that pairing has already bought most of the precision more data would.

11. What to measure when the output is prose — A

B — Treats a large movement in a number as evidence of quality, without asking what the number responds to.

C — Over-corrects into dismissing the metric entirely, when it does carry real signal about wording — just not about correctness.

D — Assumes the weakness in a metric is always statistical, when here the metric is measuring the wrong property precisely.

12. Precision and recall, worked on real numbers — B

A — Quotes accuracy on a task where 98% of items are fine, so a detector that flagged nothing would score 98% too.

C — Swaps the two formulas: dividing by the number flagged gives precision, dividing by the number actually defective gives recall.

D — Reads an F1 in the middle of the range as near-random when random flagging at this base rate would give a precision near 2%.

13. When an item has several right answers — B

A — Reads a metric artefact as a model defect; 0.92 to the fourth power is 0.72, so this is the arithmetic behaving normally rather than evidence of bias.

C — Overcorrects: exact match is the right metric wherever the product needs the entire set correct, such as auto-filling a form nobody reviews.

D — Confuses the two: Hamming loss counts individual label decisions, so it would report something near 8% here, and on a long label list it flatters a model that predicts almost nothing.

14. What a curve says, and what its one number leaves out — B

A — Reads a single arbitrary threshold as the operating point; 0.5 has no special status, and the better model at 0.5 is often the worse model at the threshold you would actually run.

C — Reaches for the metric that class imbalance ruins: at 3% spam, a model flagging nothing scores 97%.

D — Calibration is worth knowing but does not follow from AUC, and equal AUC does not mean equal ranking at the top of the list, which is the region that decides the product.

15. Moving the threshold is a business decision — D

A — Raises a genuine caveat about cross-dataset comparison that does not bear on whether this threshold is right for this system.

B — Introduces a drift concern with no evidence for it, while the stated configuration already contains a concrete problem.

C — Treats AUC as if it had an absolute pass mark, when what matters is where the operating point sits on the curve.

16. Making a 0.8 mean eighty per cent — C

A — Expects a change the method cannot produce: a monotone map preserves every pairwise ordering, so AUC is mathematically fixed.

B — Attributes a mathematical invariance to a property of this particular model, and would predict that calibration sometimes moves AUC, which it cannot.

D — Overfitting is a real risk with isotonic regression on small sets, but it would show as a worse ECE on fresh data, not as an unchanged AUC.

17. When the output is a quantity — B

A — A real effect of unit-free averaging, but it would produce errors in both directions rather than the one-sided shortfall described.

C — Names a genuine nuisance in MAPE, but dropping zero-demand days would not systematically bias forecasts downwards across ordinary days.

D — RMSE would help with large misses, but it is not the only option and it does not explain why the bias was consistently in the under-stocking direction.

18. Scoring a ranked list — A

B — Reaches for the most sophisticated ranking metric when the binding question is simply whether the answer was present in what the model was given.

C — Optimises for context tidiness, a second-order effect, while the first-order failure of a missing passage goes unmeasured.

D — Questions the measuring instrument, which is worth doing eventually, but leaves the pipeline's two halves still fused into one unusable number.

19. Three questions, not one, for a retrieval system — C

A — Buys precision on a number whose blind spot is structural, so a tighter interval around the same measurement reveals nothing new.

B — Averages two uncalibrated opinions, which smooths the number without testing anything the set does not already cover.

D — Adds operational metrics that belong in the report but say nothing about the system's behaviour on questions outside its knowledge.

20. Public benchmarks, and what they are for — B

A — Treats a small aggregate lead on academic multiple-choice as a transferable ranking for a structured-extraction task.

C — Prefers the ranking that rewards chat formatting and confident tone, neither of which is relevant to extracting fields from a document.

D — Overcorrects into dismissing benchmarks altogether, discarding their real use as a way to narrow forty candidates to three.

21. Asking what pure chance would have produced — C

A — Treats balance as the only requirement; the problem is not how many rows each side gets but which rows are treated as independently exchangeable.

B — Names a real cost of unpaired shuffling, but the more serious error here is in the opposite direction: the p-value comes out too small, not too large.

D — Confuses a data-collection question with the exchangeability assumption the shuffle encodes; the inputs here are identical and the grouping is still the problem.

22. What a p-value is, and the four things it is not — C

A — Reads alpha as the false-discovery rate; 5% is the error rate among the ideas that do nothing, not among the results that come out significant.

B — Lands near the right answer by the wrong route: the halving comes from power applied to true effects, while the false alarms come from applying 5% to the 90 null experiments.

D — Assumes correct procedure removes false positives; a 5% error rate applied 90 times produces false positives no matter how carefully each test is run.

23. Sizing the set before you run it — B

A — Spends the budget on a test with roughly a 10% chance of detecting the effect, and the interval will be too wide to support any decision either way.

C — Splitting reduces power in each test and combining verdicts by majority has no defined error rate; three underpowered tests are worse than one.

D — Loosening alpha does raise power somewhat, but it does not approach what pairing achieves here, and it raises the false-alarm rate on every future comparison run the same way.

24. The bootstrap, and the three places it lies — B

A — BCa corrects for skew in the bootstrap distribution, but it cannot conjure information about a tail that only three observations describe.

C — The general small-sample caution starts around 30 items; 300 is ample for a mean, and the specific problem here is the extremity of the statistic rather than the overall count.

D — Attributes an assumption the bootstrap was designed to avoid; freedom from a distributional assumption is its main attraction.

25. Five hundred rows, forty users — C

A — Counts rows as independent observations; conversations from one customer are correlated, so 500 rows carry far less information than 500 independent draws.

B — Abandons the estimate rather than computing it at the right level; the customer-level mean supports a perfectly ordinary interval.

D — Discarding data to reduce imbalance loses information and still treats the remaining rows as independent, which is the actual source of the error.

26. Twenty slices and one guaranteed surprise — B

A — Small slices do produce unstable estimates, but a significant result on a small slice is not automatically driven by few items, and the multiplicity issue holds even for large slices.

C — Overreaches: slice results are essential for fairness and for diagnosis, and the discipline needed is pre-registration and replication rather than a ban.

D — Gets the direction of the correction backwards; Bonferroni makes the threshold stricter, so a result significant at 0.05 across eighteen tests would almost certainly not survive it.

27. Watching the dashboard until it says yes — C

A — The test corrects for sample size but not for the stopping rule; the decision to stop was itself made on the basis of a high estimate.

B — Noise widens the spread in both directions; it does not systematically shrink estimates, and the stopping rule selects from the high side.

D — Too strong: peeking inflates the false-positive rate and biases the estimate upwards, but the data still carries information about a real effect.

28. Getting more signal from the items you already have — A

B — Averaging does reduce per-item noise, but it costs items three for one, and the arithmetic shows the mean's variance is minimised at one run per item.

C — Optimises for per-item reliability, which is the right goal for a regression suite and the wrong one for estimating an overall difference.

D — Splits the difference rather than following the arithmetic; any k above one trades items for precision the mean does not need.

29. Better in every group and worse overall — B

A — A real hazard covered later in the course, but it would be expected to move the slice-level numbers too, and here every slice held.

C — A harness bug of this kind would corrupt the slice numbers as well, so it cannot explain slices that held steady.

D — Misreads growth in a slice as newly revealing a regression; the slice rates themselves did not fall, which is what a hidden regression would require.

30. Summaries: what was invented, and what was left out — B

A — Points at the unit of comparison rather than the missing information; even a sentence-level similarity metric has the same blind spot.

C — Describes a mechanism that partly exists in BLEU, and a brevity penalty punishes shortness generally without knowing whether the missing content mattered.

D — Invents a technical limitation; the metrics compute perfectly well across different lengths, which is part of the problem.

31. Translation, and the languages nobody measures — A

B — Multiple references do help, but single-reference BLEU is standard practice and does not explain a cross-language gap.

C — Invents a requirement; sacreBLEU handles Devanagari directly, and transliterating would introduce error rather than remove it.

D — BLEU is a corpus-level ratio, not a quantity that grows with the number of segments, so size alone does not shift it in a systematic direction.

32. Code: run it, in a box, against tests it has not seen — C

A — Would justify a k above one only if the interface really samples repeatedly; the stated product shows one suggestion, so k = 5 does not describe it.

B — Confuses estimator variance with validity; the standard estimator is unbiased for any n greater than or equal to k, and 10 samples is normal practice.

D — Reaches for exactly the comparison this lesson warns against, since public benchmarks sit in the training data and describe a different codebase.

33. Structured extraction, where normalisation is most of the work — B

A — Correlation usually makes the record rate better than the independent estimate, since errors cluster in the same bad documents rather than spreading across records.

C — A real and useful check, but it moves the field number modestly; the automation claim founders on the multiplication across fields.

D — Worth measuring separately, though invalid outputs are normally counted as failures rather than silently excluded, and it does not explain the automation gap.

34. Multi-turn: score the session, not the turn — B

A — Rater context is a real effect but works in both directions, and it does not explain the specific arithmetic distortion here.

C — A rubric design flaw that some teams do have, but it is a definition problem rather than the structural double-counting the design creates.

D — Length weighting does distort the mean, though the dominant effect is the correlation between turns after a failure rather than unequal session lengths.

35. Ranking systems, where the labels are clicks and the clicks are biased — B

A — True as a general principle and not the specific mechanism; even a validated proxy measured this way inherits the position effect.

C — A genuine weakness of raw clicks that argues for a better outcome signal, but it does not explain the systematic bias created by position.

D — Showing different items is the point of the comparison; the flaw is the position-driven examination bias, not the difference in items.

36. Images: the background the model is actually reading — B

A — Missed objects hurt equally at both thresholds; a gap between thresholds is about the quality of boxes that were found, not about objects that were not.

C — Small sets do make mAP noisy, but a systematic 33-point drop between thresholds is a localisation signal rather than sampling noise.

D — Class imbalance affects the mean across classes, but it does not by itself produce a threshold-dependent gap of this shape.

37. Speech: word error rate, and the words that matter — C

A — Worth confirming, but the question states the audio is the same, and the normalisation difference explains a gap of this size on identical audio.

B — Reference quality does matter and typically shifts scores by a point or two, not by seven on well-prepared transcripts.

D — Aggregation choice does change the number slightly, but it is a much smaller effect than normalisation and is not the standard source of a gap this wide.

38. Documents: stages that multiply, and checks that need no labels — B

A — Claims more than the check can deliver: it detects a class of errors and stays silent on fields with no arithmetic relationship.

C — Treats a monitoring signal as a substitute for measurement; consistency checks cannot tell you the supplier name is wrong.

D — An arithmetic mismatch does not identify which stage failed, since a layout or extraction error produces the same symptom.

39. Tables, cross-validation, and the leak that gives you 0.99 — C

A — Refitting the estimator is not enough; the transformed features already carry information from the held-out rows before the model sees them.

B — Blames the technique rather than where it was fitted; target encoding is fine when computed inside each fold.

D — More folds change the variance of the estimate slightly and do nothing about a leak that is present in every fold.

40. LLM-as-judge, and where it misleads you — A

B — Focuses on effect size alone and misses that a genuine 68% preference over 200 inputs would be a substantial, shippable win.

C — Swaps to an absolute scale that judges use far less consistently than a forced choice between two options.

D — Proposes a reasonable-sounding review of the losses while leaving the measurement's own bias completely unexamined.

41. Turning a quality you can feel into rules a machine can apply — B

A — Assumes more scale points mean more information, when models cluster their ratings regardless of how many points you offer.

C — Names a real ordering problem that is worth fixing, but it is secondary to a score that could not be interpreted even if perfectly reasoned.

D — Assumes judging always needs a gold answer, when rule-based checks against a policy or context work without one.

42. Gold answers, and asking whether two answers agree — B

A — Attributes a systematic bias to capacity; the effect appears with capable models and short references alike.

C — A real design error when it occurs, but the anchoring effect persists even when the question is supplied.

D — Reverses the usual finding: comparison against a reference is the more stable task, provided the reference is expressed as facts rather than prose.

43. Proving the judge agrees with you, and re-proving it — A

B — Blames overall sample size when the shortage is specifically of positive cases, which stratifying fixes without any increase in labelling effort.

C — Raises an anchoring risk worth controlling for, but it does not explain why a 94% figure on this sample is uninformative.

D — Replaces the ground truth with another uncalibrated model, which measures whether two models share a bias rather than whether either is right.

44. Turning pairwise wins into a ranking you can defend — B

A — Treats a small gap from a proper protocol as decisive; 25 points is roughly a 54% win rate, which 400 comparisons cannot separate from a tie.

C — Order matters and is handled by running both orders; once that is done, the gap is interpretable, just small.

D — Correctly notes that preference is not accuracy, then overstates it: the ranking is meaningful about preference, and the problem here is the size of the gap.

45. The checks that cost nothing and catch most of it — C

A — Is the right move once the outputs are known to be well-formed, but spends judge-calibration effort on items that may have failed for a mechanical reason.

B — Escalates to a more expensive judge to confirm a number, when a free deterministic check may explain most of it in ten minutes.

D — Slicing is genuinely valuable, but it partitions a quality signal that may not yet be a quality signal at all.

46. Checking an answer against the passages it was given — B

A — Always worth verifying, but the symptom described is a consistent wrong deadline rather than scattered inventions, which points at the source rather than the detector.

C — An inference-policy issue would produce borderline cases, not a systematically wrong figure that users can identify.

D — Claim-level aggregation does flatter the number, and it would not by itself cause a specific policy figure to be consistently wrong.

47. What judging costs, and how to pay a tenth of it — B

A — Useful for the cost story and silent on whether the verdicts are still correct, which is what trusting the numbers requires.

C — Human agreement is the right check for the judge you finally rely on, and the specific question here is what the routing rule loses relative to the judge it is standing in for.

D — A practical consideration for run time, not evidence that the cheaper path reaches the same conclusions.

48. Testing what happens when somebody tries — B

A — Asks a fair question about precision that would sharpen the figure without revealing what the hardening cost elsewhere.

C — Points at a real coverage gap in the attack set, but says nothing about the collateral damage a safety improvement usually causes.

D — Tracks an operational cost that matters for shipping but not for judging whether the safety result is a genuine improvement.

49. When the output is a sequence of actions — A

B — Applies the sample-size correction correctly but treats run-to-run variance in the agent itself as though it did not exist.

C — Prescribes a genuinely useful debugging step, but reading five trajectories does not make the headline figure any more trustworthy.

D — Swaps the headline for a partial-credit measure that flatters an agent which reliably gets most of the way and never finishes.

50. The harness, and why you should build the small one first — A

B — Is a good second step, but it assumes the two runs measured the same thing, which is precisely what has not been established.

C — Chases an external cause before ruling out the far more common one that something in the run's own configuration changed.

D — Suspects the instrument, which is worth doing on a schedule, but skips the cheaper check of whether anything in the run definition moved.

51. An evaluation that runs on every change without going red at random — C

A — Retrying until success is optional stopping: it guarantees a green build regardless of quality and destroys the gate's meaning.

B — Hides a real instability by desensitising the gate, which also blinds it to the genuine regressions the band was sized to catch.

D — Deleting the evidence of instability makes the number look better and removes the signal; quarantine keeps the measurement while unblocking the queue.

52. Recording enough to diagnose a failure you did not see happen — B

A — Reverses the sizes: template plus variables is smaller, which is part of why teams choose it.

C — Token counts are normally returned by the provider and logged directly, so they survive this choice.

D — A retrospective judge needs the input, context and output rather than the template, so this is not the first thing to break.

53. Regression testing a prompt — A

B — Reads an unchanged aggregate as unchanged behaviour, which is exactly the case where aggregates are blind.

C — Concludes the change is a no-op, when it may have traded one group of failures for a different group.

D — Reaches for more data when the information needed is already sitting in the run they just did.

54. Reading fifty failures — B

A — Restricts the team to the failures that feel like AI problems, leaving the largest cluster untouched because it lives in another part of the system.

C — Assumes a prompt can address causes that sit outside the prompt, and bundles four changes so that no single one can be measured.

D — Applies sampling caution to a diagnostic exercise where the clusters are already stark enough to act on.

55. A/B testing a feature honestly — A

B — Treats a p-value as a property of the data alone, independent of the rule that decided when to look at it.

C — Names a genuine problem with short experiments that happens not to be the error being made here.

D — Assumes a stricter cutoff repairs the damage, when repeated looks inflate the error rate at any cutoff.

56. Getting evidence from live traffic without exposing users — C

A — A real operational cost that argues for sampling and time-boxing, not a safety risk to users.

B — A genuine capacity concern worth planning for, and it is a performance issue rather than the irreversible one.

D — Describes a limitation of what shadow mode measures; it is precisely why the canary stage follows, and it is not a danger created by shadow running.

57. The signals users give you without being asked — B

A — Builds the dashboard on a one-to-two per cent response rate from a self-selected group, so the level is uninterpretable and the trend is noisy.

C — Tracks usage, which can rise precisely because the feature is failing and people are trying again.

D — Scales the judge to all traffic without addressing that judge scores drift and are only meaningful as same-run relative comparisons.

58. Sampling live traffic, and keeping reviewers honest — B

A — Reads an absence of new findings as evidence of quality when the sampling method could not have produced new findings either way.

C — Blames sample size when the same 50 items drawn at random would have been capable of surfacing unknown failures.

D — Names a genuine hazard that overlap checks exist to catch, but it would not explain the absence of new failure categories.

59. The document that travels with the system — B

A — Compresses several unmeasured claims into an adjective; a reader cannot tell what was tested, on how much, or in which languages.

C — Describes effort rather than result; a card should say what the testing looked for, how much of it was done, and what it found.

D — Cites public benchmarks that may sit in training data and describe a different task, with no reference to the set that matters to the reader.

60. Buying a model you will depend on — B

A — Splits one decision into two incomparable ledgers and leaves the retry traffic out of the price entirely.

C — Sets an arbitrary bar in place of the arithmetic; whether 92% is acceptable depends on retry cost and on what a failure does downstream.

D — Leads with the measurement least connected to the task and defers the one that decides the bill.

61. Why the builder's own numbers are weak evidence — B

A — Possible and untested; treating the gap as a set-difficulty artefact removes the one diagnostic the arrangement exists to give you.

C — Overreacts: the internal set remains useful for development, and its role is to guide work rather than to produce the reported figure.

D — Label quality is worth checking, and it does not account for a 14-point gap in the direction that a builder's self-selection predicts.

62. What the emerging rules ask for, and what they do not settle — B

A — States a universal rule that does not exist; retention duties vary by jurisdiction, sector and data type.

C — A marketing use that does not require the full artefact trail and is not what the record exists for.

D — Confuses evaluation data with training data, and using a held-out set for training destroys the only thing it was for.

63. Measuring the worst case, not the average one — C

A — Wrong and immediately checkable, since the source sits next to the output, so the user detects and corrects it at once.

B — Highly visible and fully reversible; irritating rather than harmful, and a candidate for the ordinary quality backlog.

D — Low severity and easily ignored by the user, since the other four options remain visible and nothing irreversible follows.

64. Measuring the human in the loop — C

A — Would predict a low error rate before the change; the premise is that errors reaching users stayed roughly constant, which implies the reviewers were not catching most of them.

B — Plausible as a contributing factor and it does not explain why the six points of model improvement failed to translate at all.

D — A real hazard covered elsewhere in the course, and it would show as a gap between measured and observed model accuracy rather than in the oversight step.

65. Testing whether the system gives away what it was given — B

A — Repetition counts are not repeated trials of the same condition; the different counts test different conditions, which is the point.

C — Tokenisation is not what is being probed; the test is about memorisation of content, which is a property of training exposure.

D — A positive control is a useful side effect, and the substantive result is the shape of the curve rather than confirmation that the harness runs.

66. When the model is right and the product still fails — B

A — Possible and testable by scoring live samples, and it would show up as a lower measured accuracy on production traffic rather than as a drop at later funnel stages.

C — Judge leniency is worth checking against human labels, and it would move the accuracy figure a few points rather than account for a fifty-point funnel gap.

D — A reasonable challenge to the definition, and the funnel would still show where people leave, which is the diagnostic the gap points to.

67. When people disagree about what good means — B

A — A real quality-control use that treats systematic disagreement as noise, which is exactly the reading this lesson warns against.

C — A mechanical detail of the voting rule rather than a reason to preserve the underlying judgements.

D — Miscounts the statistics: several judgements of the same item are clustered observations of one item, not independent observations.

68. Cost, latency, and the model that changed under you — A

B — Rightly prefers data to anecdote, but forgets that an eval set only measures the cases someone put into it.

C — Has the right instinct about staleness but the wrong order: rebuilding now destroys the before-and-after comparison that could explain the spike.

D — Borrows a concept from interface experiments and applies it to a backend swap users never saw.

69. The tail is what your users feel — B

A — Applies a per-call percentile directly to a task made of thirty calls; the chance compounds across calls rather than staying fixed.

C — Averaging is the intuition the arithmetic contradicts: with many draws the extremes are met routinely rather than cancelling.

D — Combining is exactly what the independence arithmetic allows, and the mean would tell you less about the tail than the percentile already does.

70. Watching a system you can no longer test — C

A — Is a real possibility worth ruling out, but it would not usually shorten every output at the same time on the same afternoon.

B — Names a plausible internal cause that a retrieval hit-rate metric would show directly, and which a sudden step change fits poorly.

D — Doubts the instrument, which is worth a minute's check, but dismisses a step change of exactly the size a model swap produces.

71. The system that trains on its own output — B

A — Worth re-calibrating, and it would not explain why complaints rose specifically over a period when the corpus was changing.

C — Expectation shift is real and would not produce a specific rise in complaints about answers being factually wrong.

D — A subtler hallucination would reduce measured groundedness, since the check compares claims against the retrieved passages.

72. Refreshing a set without breaking every comparison — B

A — Publishes the right number with no way for a reader to reconcile it against every previous report, which is how a set change gets read as a regression.

C — Continues reporting a number about a population that no longer exists, which is the problem the refresh was meant to fix.

D — Blends two measurements of different populations into a figure that describes neither and cannot be reproduced by anyone.

73. Fairness as arithmetic, and the choice you cannot avoid — A

B — Treats an impossibility result as a sample-size problem, when the conflict persists at any amount of data whenever base rates differ.

C — Elevates one definition to the definition, which is exactly the move that hides the trade-off rather than resolving it.

D — Proposes a real technique for equalising odds without acknowledging that it necessarily breaks the calibration the reviewer wants to keep.

74. Turning a complaint into a permanent test — B

A — Names a real hazard that regular calibration exists to catch, but it would not by itself decouple the score from user satisfaction in this direction.

C — Trusts the offline number over the user signal without first asking whether the offline set still represents the same population.

D — Points at a labelling defect that would show up as a fixed set of always-failing items rather than a gradual drift.

75. Writing the result down — D

A — Attaches the caveat to a conversation while the number goes onto a slide that will be forwarded without it.

B — Invents a number by informal adjustment, which is less defensible than the measured one because nothing supports the discount.

C — Treats an incomplete measurement as no measurement, withholding a result that is genuinely sound for the two slices it covers.

76. How much of this to actually do — A

`B` — Buys statistical precision first, when the gap is the absence of monitoring, slicing and an incident loop rather than the width of an interval.

`C` — Applies the tier-3 programme to a reversible beta, which is a real cost that will be abandoned before it pays for itself.

`D` — Assumes failures produce complaints, which holds for visible errors and not for the quiet ones that run for months.

Module test — 1. What counts as evidence

1. B

With zero failures in twelve tries, a reasonable upper bound on the true failure rate is about one in four. Small informal samples can rule out catastrophe; they cannot separate very good from roughly okay, and the twelve items were written by the team rather than drawn from what customers send.

2. A

A threshold chosen after seeing the number is a rationalisation; every number looks like support for what you already wanted. Written beforehand, dated and committed beside the set, it is a commitment the team can be held to, including at midnight. It has nothing to do with p-value validity, and stopping early once a threshold is reached is the optional-stopping error the statistics module warns about.

3. D

A set-wide rate on a stratified set describes a world that does not exist. Either report per stratum and never average, or weight each stratum back to its production share. The weighted number rests on few items in the rare strata, so its uncertainty is wider than a count of items implies, and that should be said when it is quoted.

4. C

This is the kappa paradox. When one class dominates, expected agreement approaches observed agreement and the ratio becomes unstable. Kappa is asking how much better than guessing at these rates, and guessing is already 94% right. Report raw agreement, kappa and both marginals together, and on very rare classes prefer agreement measured only on the items either person flagged.

5. B

Synthetic items answer whether a system can handle a shape of input. They cannot say how often that shape occurs or how often it fails in traffic, because frequency is a property of your users and no generator has access to them. Generated items belong in a capability suite, tagged by source, and the harness should refuse to average them into the headline.

6. D

Reading failures and changing the prompt to fix them, twenty times over, is overfitting with a human as the optimiser. The dev score becomes a record of how well a hundred specific examples were fitted. A test set that is looked at rarely is what makes the gap visible; a leak would make the test score look better, not worse.

Module test — 2. Metrics that mean something

1. C

Precision is true positives over everything flagged: 48 of 80, which is 0.60. Recall is true positives over everything that should have been caught: 48 of 60, which is 0.80. Both come from the same four cells, and the arithmetic should be doable on paper.

2. A

Accuracy is (TP + TN) over everything, and under imbalance the true negatives dominate the numerator. The rarer the thing you care about, the more accuracy measures the ability to say no. Whenever accuracy is reported on a rare event, ask for the base rate; without it there is no result.

3. B

The most useful shape in practice is a band with a human in the middle. Below the low edge, act as negative; above the high edge, act as positive; between, send to a person. Widen or narrow the band until it matches the review capacity. It gives high precision on both automatic paths and spends scarce human attention where the model is unsure, and it degrades gracefully when volume spikes.

4. C

Platt scaling, isotonic regression and temperature scaling are all monotone maps from raw score to probability, so every pairwise ordering is preserved and AUC, average precision and every recall-at-precision figure stay exactly where they were. Calibration makes the number usable as a probability; it cannot make a weak model strong.

5. B

MAPE explodes near zero, so a quiet night or a slow-moving product dominates the score, and its asymmetry biases model selection towards forecasting low. WAPE sums absolute errors and divides by the sum of actuals: unit-free, no explosion, weighted by size, which is usually what the business means by how far off were we. Squaring is RMSE; the naive comparison is MASE.

6. D

If the supporting passage is not in the retrieved context, no prompt, model upgrade or instruction can produce a correct grounded answer; the model can only decline or invent. Recall at the k you actually pass in is a hard ceiling on answer accuracy, and measuring it first ends about half of these investigations on the spot.

Module test — 3. Deciding whether a difference is real

1. C

What you shuffle is your claim about what is exchangeable. Rows from the same customer resemble each other, so shuffling rows treats correlated observations as independent and gives a confidently wrong p-value. Shuffle at the level that is actually independent, the customer, carrying all of each customer's rows together.

2. B

Of the 10 real improvements, 50% power detects 5. Of the 90 that do nothing, 5% come out significant anyway: 4.5. So about 9.5 wins, nearly half of them noise, with every test behaving exactly as advertised. The false-discovery rate depends on how often your ideas are right, which is why raising power and replicating surprises matter more than lowering alpha.

3. D

The sample size is roughly 16 p(1 – p) / d². Halving d quadruples n: 1,024 becomes about 4,100 per group. This is why most hundred-item comparisons cannot support the claims made from them, and why pairing, which counts only discordant items, is the single largest efficiency available.

4. B

The largest possible bootstrap maximum is the observed maximum, so an interval for a tail is bounded by your data rather than by the world. With 200 items p99 is computed from two observations; the interval looks tight and the next 200 requests contain an outlier the sample never saw. Do not bootstrap a percentile beyond roughly 1 – 5/n; get more data or model the tail.

5. A

Crossing the threshold means the estimate is unusually high at that moment, so the effect you stop on is drawn from the upper part of its own distribution. This is the winner's curse: a four-point gain declared on day three becomes 1.5 points when the experiment runs out. Treat an early-stopped effect as an upper bound until it is re-measured.

6. B

Both overall figures are weighted averages with different weights: 500/500 for the old model, 800/200 for the new one. Hindi is harder, so more Hindi pulls the average down even though every slice improved. Report per slice, and if one number is required, standardise both systems to a fixed reference mix and version the weights.

Module test — 4. The systems you will actually be handed

1. C

Claim-level percentages flatter the system, because most claims in any summary are uncontroversial scaffolding. The unit a reader experiences is the summary, and a summary with one invented date is a wrong summary. Report the share of summaries with at least one unsupported claim as the headline, and weight by consequence where you can.

2. B

pass@k estimates the chance that at least one of k samples works. If the person sees one suggestion, pass@1 is the product's number; pass@5 describes a tool that shows five options or an agent that retries against a failing test. Papers quote the flattering one; decide yours from the interface and say which you used.

3. D

Any literal the model extracted should appear in the source after normalisation. A one-line substring check catches the invoice number that was never printed, without a model call and without a label, which means it can run on every document in production. Computed fields, such as a total the model summed, need a short explicit exception list.

4. A

Scoring turns individually records eight failures from one mistake, in rows that are anything but independent, and makes one breakdown look like a widespread quality problem. Score the session, and record the first wrong turn: a pile-up at turn one is comprehension or routing, around turn twenty it is the context window.

5. B

The classic vision leak is the photo session: images taken in one place with one camera in one minute share background, lighting and framing, and a random split lets the model learn those. Split by whatever unit generated a batch of similar images, run a near-duplicate check, and test with the correlation deliberately broken.

6. C

An implausible score on a business problem is a bug until proven otherwise: a column that could not exist yet, preprocessing fitted outside the fold, duplicates across splits, or an identifier that encodes the answer. The leaking column is almost always at the top of the importances, and the whole investigation takes about fifteen minutes.

Module test — 5. Automating judgement

1. D

The rate at which the two orderings disagree is the judge's noise floor. Running every pair twice with the order swapped and counting a win only when both agree is the cheap fix for position bias, and it also tells you which margins are real. A 25% self-disagreement rate does not make the judge useless; it sets the scale.

2. B

Given all six rules at once, the model decides the reply looks fine or looks bad and produces six judgements consistent with that impression; outputs cluster on all-pass and all-fail far more than independent rules would. Six calls cost six times as much and buy six independent pieces of evidence, which is cheaper than being wrong.

3. A

Every gold set carries errors: a policy that changed, a mislabelled item, an annotator's misreading. Five to ten per cent is normal, and it scores a model that is right where the reference is wrong as failing. At scores in the nineties most remaining failures may be the file, so audit the item rather than the model, correct the references, and note the correction rate.

4. C

Elo is an online approximation of the same model, built so chess ratings could be updated by hand between tournaments. Its order-dependence and step-size sensitivity are exactly what an evaluation does not want. With all the comparisons in hand, a logistic regression gives the strengths with standard errors for free, and a bootstrap over comparisons puts an interval on every rank.

5. A

Schema validity, forbidden strings, grounded numbers, resolving citations and length bands cost microseconds and decide a large share of items with no ambiguity. Every check moved down into code is a permanent saving and a permanent gain in reproducibility, and a judge asked about a truncated output has spent a call to learn what json.loads knew for free.

6. B

Agents branch, so a single run teaches almost nothing and the distribution across runs is the finding. Three of five is not a passing task, and reporting it as one is how demos become incidents. Report the mean pass rate alongside the number of tasks that passed every time; the second is usually the one a business would rely on.

Module test — 6. Evaluation in the working loop

1. B

A run is identified by everything that could change the answer, and the dataset hash matters as much as the git sha. Somebody adds forty items, the name stays support-v4, and two runs on different sets are compared as if they were the same. Record the hash, the prompt version, the model version and the seed with every run.

2. C

Re-running a failing item until it passes is optional stopping with a CI badge. Retries are for transport failures only. Items that genuinely flip move to a quarantine list that is still run and reported but does not block, and a growing list means the prompt is unstable or the rubric has an ambiguous case, both worth an hour.

3. D

Not the template: the rendered string or message array actually sent. Every production failure is diagnosed from what was recorded, and a complaint from Tuesday cannot be reproduced once the corpus and the prompt have moved on. Versions, retrieved chunks, tool calls, finish reason and what the user did next complete the minimum record.

4. B

An aggregate can hold perfectly still while a third of the verdicts change places. Seven items in and seven out, with landmark cases falling from 61% to 44% while street addresses rose, is a trade that might be correct but should be a decision. Print the flips and the per-tag breakdown; the total is for the meeting.

5. A

Only the first page of the receipt was in the input is a cause; extraction error is a category that would have hidden it. Write fifty lines, then sort and merge, then count. The clusters that emerge are usually not what the team assumed, and the biggest is often a data or product problem rather than a prompt problem.

6. C

A shadow run reads real inputs and must discard its outputs: it must not send the email, write the record, call the payment API or post the message. Enforce it with a flag the tool layer checks, and verify it against a staging endpoint that asserts nothing arrived. Every team that relies on convention learns this from a customer's second copy of an email.

Module test — 7. Evidence other people can act on

1. A

Performs well in Hindi is marketing with a technical typeface. Hindi: 84% task success (95% CI 78 to 89) on 300 items sampled from March traffic, labelled by two reviewers at kappa 0.71 is a card. Where something was not measured, the card says so, and the out-of-scope section is specific enough to stop somebody.

2. B

Two attempts double the spend per task, and 8% malformed output lowers the denominator, so B is not cheaper per task that actually succeeded. Cost per successful task is the number to insist on, then express it per business unit, per resolved ticket or per document, because that is what will be compared against the alternative.

3. C

Independently verified frequently means a consultancy ran the vendor's own test suite. Three questions extract the difference: did the evaluator choose the set, was the protocol fixed before results were seen, and was publication guaranteed regardless of outcome. If any answer is no, it is a review, and those are also the three commitments to make when you commission one.

4. D

An error a user can spot is a nuisance; the same error where verification is impossible is a harm. Harm concentrates exactly where the person cannot check the answer, which is why showing the source beside the answer, a draft instead of a sent message, or a refuse-and-route response can reduce harm without touching the model.

5. B

The miss rate cannot be computed from ordinary traffic because you do not know which items were wrong. Seeded errors are proficiency testing, used for decades in radiology and airport screening. Tell reviewers in writing that seeded cases exist, seed realistic errors, and use the results to design the system, never to discipline an individual.

6. C

The bug has a specific shape: the tenant filter runs after ranking, or in the application layer where one code path forgets it, and everything works in testing because the test data has one tenant. Build two tenants with unique nonsense strings, query from one for the other's content, and assert nothing crosses, on every change.

Module test — 8. Living with it

1. A

Generation time is right-skewed by construction, so the mean sits between the median and the 90th percentile and describes nobody. The user at the 95th percentile is the one who leaves, and a ten-call session meets that percentile about 40% of the time. Report p50, p95 and p99, and the per-session total.

2. B

Refusal rate is the single most sensitive detector of a silent model change, and it moves before anybody complains. A step change with nothing deployed is a provider-side event, and a pinned version string plus a twenty-item canary set run hourly is how you know within an hour rather than from a customer in three weeks.

3. D

Groundedness measures faithfulness to the corpus, and once the corpus is partly the model's own output the check has gone blind. Provenance metadata on every document, a policy that machine-authored material is not grounded against, and a frozen human-authored set run quarterly are the defences; detectors of generated text are not.

4. A

A number that moved because the set moved looks exactly like a quality change. Version the set with a date, keep a frozen core for like-for-like comparison, and report old and new for one cycle on the same system; the six-point gap is then the most informative number in the refresh rather than a phantom regression.

5. C

Kleinberg, Mullainathan and Raghavan, and Chouldechova, showed independently that with unequal base rates calibration and equalised odds are incompatible unless the classifier is perfect. Fairness is therefore a definition you choose according to the harm you most want to avoid, written down with the reason, with the other definitions reported so the trade-off is visible.

6. B

Each incident item was chosen because the system failed it, so a set dominated by them is systematically hard and unrepresentative; every number is pessimistic and the team stops believing it. Two files, two purposes: a regression suite where every item must pass, and an eval set refreshed from traffic that estimates quality on the work you actually do.

Knowing If It Works: end-of-course examination

1. C ***1. What counts as evidence***

A score is only readable between a baseline and a ceiling. Written out, the result is trivial 61, keywords 74, model 78, humans 88. The model's real contribution over something twenty lines long is four points, at a per-call cost and nearly a second of latency the keyword rules do not carry, and that comparison is where the decision is actually made.

2. A ***1. What counts as evidence***

A leading metric on a fixed set is a claim that it predicts the lagging one, and that claim should be checked at least once. When the offline number moved and the real one did not, the team has learned that its leading metric is not measuring what matters, which is more valuable than any single result and the reason to keep both.

3. D ***1. What counts as evidence***

Near-duplicates are the silent inflater: four hundred copies of one automated message look like a set of four hundred and behave like one item. If any normalised string exceeds about 2% of the set, deduplicate to one copy before labelling, and record its frequency, because that frequency is exactly what weighting back to traffic needs.

4. B ***1. What counts as evidence***

Boundary rulings are how a criterion survives contact with data nobody imagined. The cases you argued about, and the decision you made even when it was arbitrary, are what let a new labeller produce the same labels next quarter. A five-point scale does the opposite: it becomes each person's mood.

5. C ***1. What counts as evidence***

If the labels are themselves 92% correct, a perfect model scores 92% and a model that has learned the labellers' mistakes scores higher than one that is right. Teams routinely find a third of the failures near a plateau are label errors. No prompt climbs above the quality of the ruler, so check the ruler first.

6. A ***1. What counts as evidence***

Only the discordant items carry the comparison. The rule is that the difference is worth believing when $|b - c|$ exceeds $2\sqrt{(b + c)}$. Here that is 6 against 6.3: nearly, but not yet. With $b = 1$ and $c = 7$ the same six-point gap would clear 5.7 and be a result. Which items moved matters more than how many.

7. D ***1. What counts as evidence***

If what you build runs on next month's inputs, evaluating on a random sample of last year's is optimistic: vocabulary, product names, scam patterns and customer questions all drift. A time split also exposes features computed with information that would not exist at prediction time, which a random split rewards. Lower and honest beats higher and wrong.

8. B ***2. Metrics that mean something***

Micro averaging weights by frequency, so it is dominated by the big classes and equals accuracy in the single-label case. Macro averaging treats an eight-item class like an eight-hundred-item one. Both numbers are true; only one of them tells you the sales queue is broken, which is why the per-class breakdown is printed before any average.

9. D ***2. Metrics that mean something***

0.9 to the fifth power is 0.59. Exact match punishes brutally by construction. Start from per-label precision and recall to see where failure concentrates, use sample-averaged F1 for what a user experiences, and quote exact match only where a human will not review the whole record.

10. A ***2. Metrics that mean something***

AUC is the probability that a random positive outranks a random negative. Count the pairs: 0.9 beats all four, 0.6 beats three, 0.4 beats two, nine of twelve. No curve is required, and knowing that AUC is this pair count explains why it ignores calibration, averages over thresholds you will never use, and barely notices false positives when positives are rare.

11. C ***2. Metrics that mean something***

Under heavy imbalance the ROC curve is misleading because its x-axis divides by all the negatives: 121 extra false alarms move it from 3.4% to 16%. Precision divides by the flagged count, so it feels every false positive, and the review queue that has grown fivefold is visible on the precision-recall curve. For rare positives, read that curve and quote average precision.

12. B ***2. Metrics that mean something***

A number the model types in prose is text, produced the same way as the rest of the text, and it clusters near 90 regardless of correctness because that is what confident answers looked like in training. Where log-probabilities for the answer tokens are exposed, they behave sensibly and can be calibrated; where they are not, get uncertainty from agreement across several samples.

13. D ***2. Metrics that mean something***

MASE divides the model's mean absolute error by the naive method's error on the same data. At 1.0 the model does exactly as well as tomorrow equals today, or this Tuesday equals last Tuesday. Below 1.0 it is better than repeating the past; above 1.0 the business would do better deleting it.

14. A ***2. Metrics that mean something***

Every RAG set is built from questions the corpus can answer, so nobody finds out what the system does otherwise until production, where it is asked constantly. Add twenty unanswerable items and score abstention separately. A system that answers well and refuses badly produces confident, plausible, invented answers, and users trust a citation more than prose.

15. D ***3. Deciding whether a difference is real***

Two copies of the same system should produce an unremarkable gap and a p-value scattered anywhere from 0 to 1. A significant A/A result is nearly always one of five bugs: different subsets after an item errored, results compared in the wrong order, a cache serving one run, an ordered set truncated differently, or non-determinism larger than assumed, which is a finding. One extra run, and it has caught bad harnesses at almost every team that bothered.

16. B ***3. Deciding whether a difference is real***

A p-value is the probability of a result at least as extreme as the one observed, assuming the null hypothesis is true. It is not the probability the null is true, which needs a prior the p-value does not contain, and it says nothing about the size or importance of the difference. Report the effect and its interval beside it.

17. C ***3. Deciding whether a difference is real***

Not significant does not mean no difference; it means the experiment could not tell. The interval supports a decision where the p-value does not: a serious regression is ruled out, a large gain is ruled out, and if both would matter the experiment was the wrong size. Re-running until significance is optional stopping.

18. A ***3. Deciding whether a difference is real***

The design effect is 1 + (m − 1) × ICC, here 1 + 11.5 × 0.3 = 4.45, and 500 divided by 4.45 is about 112. The honest interval is a bit over twice as wide as the one computed on 500 rows, which is precisely the discrepancy a team sees when each fresh sample lands six or seven points away. Aggregate to the customer before computing anything.

19. D ***3. Deciding whether a difference is real***

Benjamini-Hochberg compares the i-th smallest p-value against (i / m) × q. Rank 1 is tested against 0.005 and passes; rank 5 against 0.025 and fails. Bonferroni would reject both. The gain is real and bought by accepting that about one accepted lead in ten is noise, which is fine for a list of leads and not for a shipping decision.

20. B ***3. Deciding whether a difference is real***

Subtracting the covariate's influence leaves the expectation unchanged and reduces the variance by a factor of $1 - \rho^2$. At $\rho = 0.7$ that is 0.51: the same precision as doubling the number of labelled items, for eight lines of code. Offline the covariate can be a baseline's score, a difficulty rating, or the previous release's result on the item.

21. C ***3. Deciding whether a difference is real***

With n items and k runs the variance of the mean is (item variance + run variance / k) / n, and for a fixed n × k it is minimised at k = 1. Repeats buy precision on the individual item, which is what a regression suite or a customer's reported bug needs. Which system is better overall is a different question with a different design.

22. A ***4. The systems you will actually be handed***

BLEU counts whole-word n-gram matches against one reference, so morphologically rich languages lose points for being right in a different inflection. chrF handles inflection and languages without spaces far better. Neural metrics correlate better still but were trained mostly on high-resource pairs, so their reliability drops exactly where help is needed most.

23. C ***4. The systems you will actually be handed***

Simulated users are cooperative: they answer the question asked, do not go silent, rarely misspell, and do not change goal halfway, so their success rate runs several points above reality. Calibrated against real transcripts, the simulator is a cheap way to compare versions. Write both numbers in the report.

24. B ***4. The systems you will actually be handed***

The gold file says 2026-03-31 and the model says 31/03/2026; both are right and a naive comparison scores both wrong. Most of a first extraction evaluation measures the comparison code. Report exact and normalised accuracy as two numbers, normalise per field type rather than globally, and treat the gap as the cheapest thing on the list to fix.

25. D ***4. The systems you will actually be handed***

Merging both rankings into one list and attributing each click to the ranker that contributed the item means every user compares both systems, and the between-user variance disappears. It typically needs one to two orders of magnitude less traffic, works only for ranked or multi-option outputs, and measures preference between rankings rather than any absolute quality.

26. A ***4. The systems you will actually be handed***

Word error rate is meaningless without the normaliser that produced it. None of the normalisation choices is wrong; publishing the score without them is. Use a published normaliser, or version your own with the results, and apply the identical one to every system compared, including the vendor's.

27. C ***4. The systems you will actually be handed***

Documents carry redundancy, and an arithmetic failure means something was misread even though nobody knows the truth. These checks need no gold label, so they run on live traffic, and their failure rate moves the moment a capture setting or a supplier's template changes. Log it as a production quality metric; it costs nothing per document.

28. B ***4. The systems you will actually be handed***

Plot the score against training-set size with the same folds. A curve still climbing at 100% says more rows are the cheapest improvement available. A curve that flattened at 40% says the opposite, and this one plot has redirected more projects usefully than any model comparison, at the cost of a loop over five sample sizes.

29. C ***5. Automating judgement***

Judges score longer, more structured, more headed answers higher, largely regardless of content, and the easiest way to change a prompt is to make it produce more words. If judge score correlates strongly with length, the quality improvement may be a verbosity improvement, and users may hate it. Measure accuracy separately from preference.

30. A ***5. Automating judgement***

Fail a gate and the item fails regardless of the rest, because a fabricated refund date is not offset by good structure. Build that into the score: report gates as a separate raw count, R1 violated in 3 of 200, never blended into a percentage. An average across rules is exactly the shape of number that hides the failure the product cannot tolerate.

31. D ***5. Automating judgement***

Rubrics written at a desk are neat and wrong; the ones that work come from reading fifty real failures and asking what rule would have caught each. A rule that never fires is either the wrong rule or a solved problem, and either way it contributes cost and nothing else. Version the rubric, because retiring or adding a rule changes what the score measures.

32. B ***5. Automating judgement***

Judge quality is per-class recall, not overall agreement, and a random 50 gives two examples of the class that matters. Stratify the calibration set deliberately, keep it frozen and separate from anything used for tuning, and remember when reporting that it was stratified. It costs two people about five hours, once.

33. C ***5. Automating judgement***

Grounded is not true. Groundedness measures whether the answer follows from the retrieved passages; if the passage is out of date, a perfectly grounded answer is false. It pairs with source quality, which is a corpus problem of freshness and provenance, and with answer correctness against reality on a labelled subset, and it replaces neither.

34. A ***5. Automating judgement***

A judge's verdict depends on the input, the output, the rubric and the judge version, so cache on exactly those. A change that touched no outputs then costs nothing, and in day-to-day development a well-keyed cache removes 60 to 90% of calls because most edits change a few outputs rather than all of them. Cache the claim decomposition separately, since claims survive rubric changes.

35. D ***5. Automating judgement***

The attack that matters for anything with retrieval, browsing or tools is not in the user's message. It is inside a document, a web page, an email or a database row the system reads: white text, a code comment, a footnote, a claim to come from the administrator. Put a small corpus of poisoned documents in the test index and measure how often the system follows them. The correct rate is zero and the observed rate rarely is.

36. B ***6. Evaluation in the working loop***

Transient errors, 429, 500 and timeouts, are retried by the client with backoff and counted separately, because scoring them zero drops the number for reasons unrelated to the system, and dropping them silently makes two runs cover different subsets, which is one of the five A/A bugs. Never retry a bad answer; retrying until something passes is not evaluation.

37. D ***6. Evaluation in the working loop***

Compare against a rolling baseline from recent runs on the main branch, and fail only when the drop exceeds max(3 × run-to-run standard deviation, a minimum band). A gate whose sensitivity is derived from data survives arguments; a gate set at a round number is raised, retried around and eventually disabled.

38. A ***6. Evaluation in the working loop***

A reply that gains a heading and a sign-off passes every check and annoys users long before it moves a score. A reviewer catches it in one second in a diff. Golden outputs in the pull request are the habit that makes that visible; a hard gate on a hundred-item set mostly produces false alarms that teach people to stop reading.

39. C ***6. Evaluation in the working loop***

Regeneration rate is the best free signal most products already log, and it is a proxy until checked, because people sometimes regenerate for style. Take a week of traffic, compute the signal and label quality on the same hundred sessions, and see whether they agree. If they do, you have earned the dashboard; if not, you have learned that cheaply. The same sample yields new eval items.

40. B ***6. Evaluation in the working loop***

Review only the targeted sample and you learn about the failures you can already detect, a closed loop that congratulates itself. Twenty random items a day is the only unbiased error rate and the only source of unknown failure kinds. It is boring, so it is the first thing cut. Protect it, and count how many new failure kinds each sample produces.

41. D ***6. Evaluation in the working loop***

Four hundred sessions a week will not detect a three-point change this quarter or next. An underpowered test produces a number, and a number outranks a hunch in most meetings even when the hunch is better evidence. Say so, and use interleaving, a long boring baseline, or ten real conversations instead.

42. A ***6. Evaluation in the working loop***

The canary share is chosen by arithmetic: how many events are needed before the guardrail could trigger. Five hundred requests a day at one in a thousand is one event a day, far too slow to stop anything. Either raise the share or pick a guardrail with a higher base rate. Keep assignment sticky and check the canary population is not one region or platform.

43. D ***7. Evidence other people can act on***

Most teams end with two cards, and the external one may be shorter. The moment the published card says something the internal one disproves, the team has a document that will be read back to it. Keep the card in the repository beside the code, and make a model change, a prompt change that moves the numbers, or a new language a trigger to update it.

44. B ***7. Evidence other people can act on***

A supplier change will come: a price move, a deprecation, an outage, a policy change. An untested fallback is a hope. A quarterly run on a second provider, with the result kept, turns it into a week rather than a quarter, and it is the only way to notice that prompts have overfitted to one provider's quirks, which they do gradually and without anyone deciding to.

45. C ***7. Evidence other people can act on***

A team evaluating its own work stops when the number is good, fixes what it finds so the set stops containing it, and writes a rubric that asks about what the prompt was designed to do. None of that is fraud, and all of it makes the number too high in a direction nobody can quantify. A set the builders never see is how the size of that effect becomes visible, and it is one of the most valuable numbers a team will compute.

46. A ***7. Evidence other people can act on***

There is no way to have both verifiability and freedom from contamination. Publishing the design and a sample lets a reader judge the method and reproduce it on their own data; holding the rest and rotating it limits the life of any leak through use. Say which parts are which, so a reader knows what they are relying on.

47. D ***7. Evidence other people can act on***

Every framework converges on documented intended use, data provenance, per-group measurement, logging, human oversight, incident response and periodic re-evaluation with results retained. That last item means keeping the set as it was, the results, the model version and the report for a version shipped eighteen months ago, not just the current ones. Storage is cheap; reconstructing a defence is not.

48. B ***7. Evidence other people can act on***

Near misses are far more numerous than incidents, they cost nothing to collect because the guardrail already fires, and they move first. A doubling after a change is a finding even with no harm delivered. Track the rate over time and treat a rise as the leading indicator it is.

49. C ***7. Evidence other people can act on***

Model accuracy is a component measurement; the product measurement is a funnel. A correct 300-word answer with no link and no source loses people between correct and acted on, and a rubric that scores completeness cannot see it. Pair the offline set for attribution with the funnel for consequence, and use an existing product event for resolution rather than a survey nobody answers.

50. A ***7. Evidence other people can act on***

Annotators' backgrounds predict their labels, and majority vote encodes the dominant group's interpretation as truth; every later number inherits that choice invisibly. Storing per-annotator labels costs a column and preserves every option: you can aggregate later, report agreement and composition, and score the system against each group's labels where they diverge systematically.

51. A ***8. Living with it***

Quality is a row, not a number: pass rate, p50, p95 and cost printed together, because a trade is only visible when the numbers sit side by side. Convert cost into the actual monthly bill in the currency you pay in; at 100,000 calls a day the cheap model is the difference between roughly $10,500 and $1,050 a month, and only that figure is a decision.

52. C ***8. Living with it***

Long, hard, more-likely-wrong requests are the ones that run out of time, so the answers you measured are the easy ones that finished. Count timed-out requests as failures in every quality metric and report the timeout rate beside every quality number; a change that gains two points while doubling timeouts has made the product worse.

53. B ***8. Living with it***

Traffic has a daily and weekly shape, so comparing to an hour ago produces an alert every Monday. Compare to the same weekday-hour over the last four weeks, require a minimum number of requests before a rate counts as a signal, and alert on three consecutive windows outside the band rather than a single spike. An alert that fires falsely on a schedule trains people to ignore the real one.

54. D ***8. Living with it***

Acceptance is a useful and biased signal. Accept-as-gold pipelines contain the model's successes and its undetectable errors and exclude its obvious ones, so retraining sharpens exactly the failures users could not see. Sample what was rejected and edited, and label a random sample on a fixed schedule regardless of what the user did next.

55. A ***8. Living with it***

Sum the absolute differences in bucket proportions and halve it. Under 0.1 the set resembles traffic; around 0.2 the headline is drifting; above 0.3 you are reporting on a different population. At 0.27 the refresh should be scheduled, with the set versioned, a frozen core kept, and both versions reported for one cycle so the change is not mistaken for a regression.

56. C ***8. Living with it***

An average rewards work on the majority group. A rule that no slice below a stated level ships forces work on the slice that is worst, which is usually one that was never evaluated. Tag every item with the slices you can observe, require at least 30 items per slice or say you cannot tell, and publish the table worst slice first.

57. B ***8. Living with it***

Fourteen items carry an interval of about 25 points either way. Reporting 64% invites a comparison with another number that has no interval. Reporting that the number could be anywhere from 40% to 88% and the only fix is more items buys more credibility than five confident figures, because it shows the confident ones mean something. Do not hide the slice; say what it cannot tell you.

58. D ***8. Living with it***

Twenty items and deterministic checks are right for an internal drafting helper and wrong for a support assistant in front of customers. The common failure is not neglecting a tier-three system; it is that something started as a prototype, went in front of customers, and kept its tier-one process because nobody re-asked who is harmed, whether it can be undone, whether anyone would notice, and how many people how fast. Re-ask at every launch.