READ MODE · ANSWER FIRST

How to test non-deterministic LLM systems with evals

Learn datasets, deterministic checks, RAG metrics, LLM judges, human review, repeated trials, variance, and production regression gates.

CHEAT SHEET · 6 ESSENTIAL IDEAS

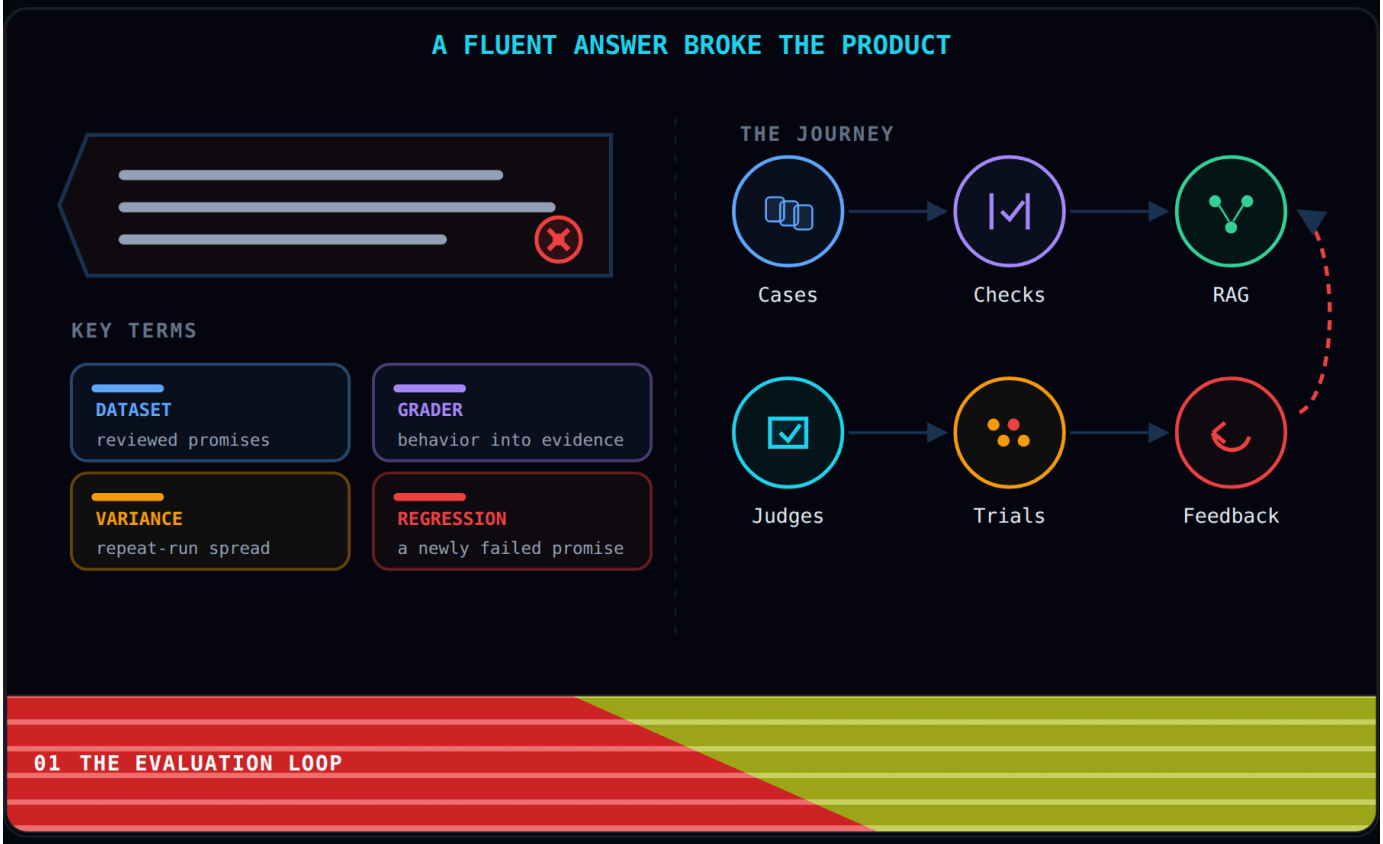
The whole story in 6 lines

Reliable LLM testing layers exact contracts, graded behavior, repeated evidence, and production feedback around a versioned dataset.

1. A versioned dataset turns product promises and real failures into repeatable cases.
2. Program checks should protect schemas, tool arguments, permissions, and observable effects exactly.
3. Retrieval, grounding, and tool metrics locate failures in different parts of a RAG workflow.
4. Model graders scale judgment, while human labels expose bias and keep the rubric tied to product needs.
5. Repeated trials reveal average quality and spread, so one lucky response cannot decide a release.
6. Production signals become new cases, closing the loop between live failures and future regression tests.

STAGE 1 OF 8

The Evaluation Loop



A support assistant confidently promised free return shipping, but the company policy never said that. The wording sounded excellent while the behavior failed a real product promise.

A dataset stores reviewed situations we care about. A grader turns each expectation into a repeatable score or decision, which gives the team evidence beyond a polished demo.

Variance describes normal differences across repeated runs. A regression is a new failure against a behavior that previously passed, so these ideas separate ordinary movement from harmful change.

We will move from reviewed cases through exact checks, RAG evidence, calibrated judgment, repeated trials, and live feedback. Now let us start with the dataset that makes every later comparison fair.

Datasets and Goldens

02 DATASETS AND GOLDENS

We begin with four reviewed support cases, not a giant generic benchmark. Each card names a realistic input, the behavior the product needs, and the kind of failure it represents.

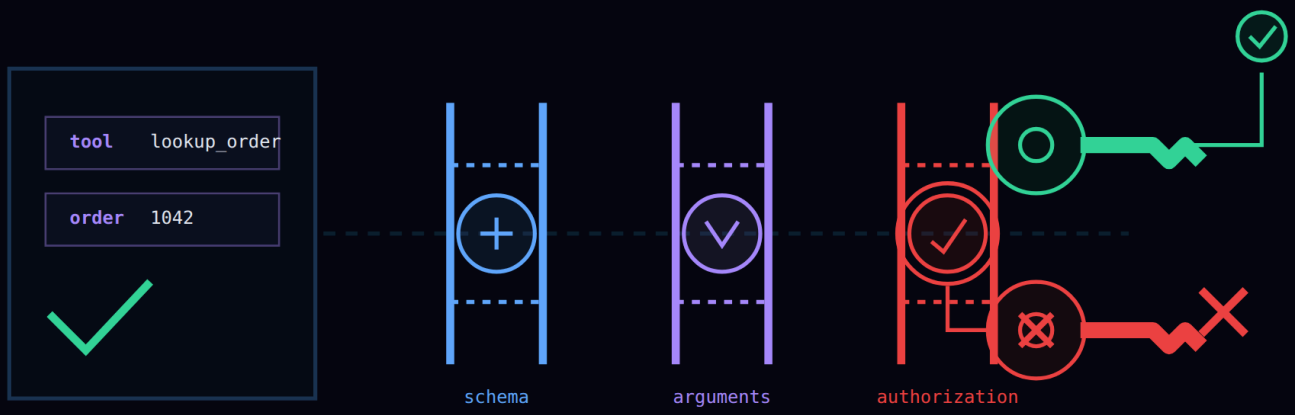
Typical cases protect the common path, while edge and adversarial cases probe places where a helpful answer can become unsafe. Production logs can suggest cases, but people still review what the expected behavior should be.

The baseline and candidate now receive the same case cards. Holding inputs and expectations fixed makes the changed prompt or model the important experimental difference, which keeps the comparison honest.

Three promises remain green, but the unauthorized refund case falls from pass to fail. Because that exact case passed before, the red move is a regression rather than vague dissatisfaction with the new system.

A useful golden set is small enough to review and broad enough to expose product risks. Next, we will turn the machine-readable parts of those promises into exact program checks.

Deterministic Program Checks



03 DETERMINISTIC PROGRAM CHECKS

The dataset tells us what matters, so we can now choose the strictest reliable checker for each property. Machine contracts should not be delegated to another uncertain model in production.

The proposed tool call first passes through a schema check. Required fields and types are exact facts, which means ordinary validation code can reject malformed output without debating its meaning.

Next, code compares the tool name and order identifier with the expected action. A valid JSON object can still target the wrong order, so structure and argument correctness remain separate checks.

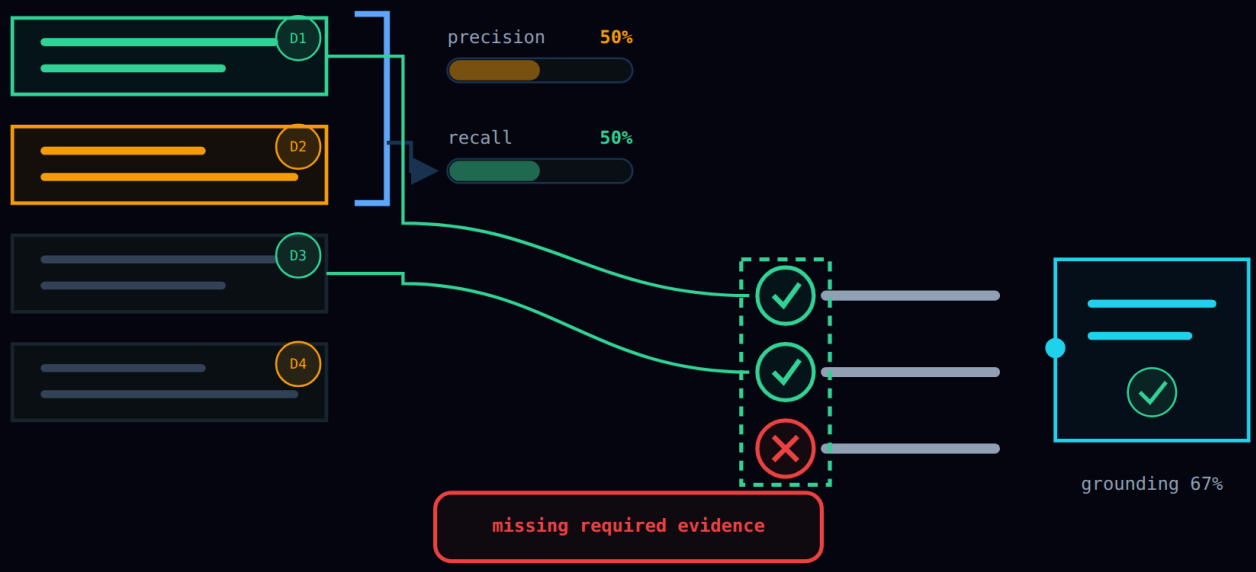
Authorization blocks the refund proposal before an executor receives it, while the safe lookup continues to an observed result. The decisive evidence is what the application allowed and what actually happened.

Exact code checks protect schemas, arguments, permissions, and side effects. Next, we will inspect retrieval and generation separately because one final answer score cannot reveal which component failed.

STAGE 4 OF 8

RAG and Tool Metrics

Try it in Watch mode. Choose the smallest Top K that includes both required policy passages.



04 RAG AND TOOL METRICS

Now that exact contracts are protected, let us diagnose the probabilistic RAG path. We will inspect ranked documents, answer claims, and the tool result as three related but distinct outputs.

Retrieval precision asks how much returned context is relevant, while recall asks how much required evidence was found. A short list can look precise and still omit the rule needed for a complete answer.

The first two documents contain one useful policy and one distractor. If the answer needs two policy passages, will this context support a complete response?

PAUSE AND PREDICT

Will Top K of 2 retrieve every required policy passage?

- **Yes, both are present**
- **No, one is still missing**

Skip this check

The context window now extends to the third document, which completes retrieval recall despite one distractor. Claim links then expose the unsupported shipping promise, while the validated lookup tool records its observed result separately.

Try every Top K option and compare the outcome under the context window. Find the smallest setting that includes both required policies without pulling the fourth distractor into the model input.

Retrieval metrics test evidence selection, grounding tests claim support, and tool checks test behavior. Next, we will grade open-ended quality with a model judge that remains accountable to human labels.

STAGE 5 OF 8

Graders, Judges, and Humans

05 GRADERS, JUDGES, AND HUMANS

RAG metrics explain system components, but qualities such as clarity and helpfulness still require judgment. We begin with a written rubric so every grader evaluates the same product promise.

Program graders handle exact checks quickly, while a model judge can classify or compare open-ended responses at scale. Pairwise choices are often easier to define than asking for an unanchored quality score.

The judge is another model, so its labels are measurements rather than ground truth. Teams should test order sensitivity, provide references when available, and keep a reviewed human sample for calibration.

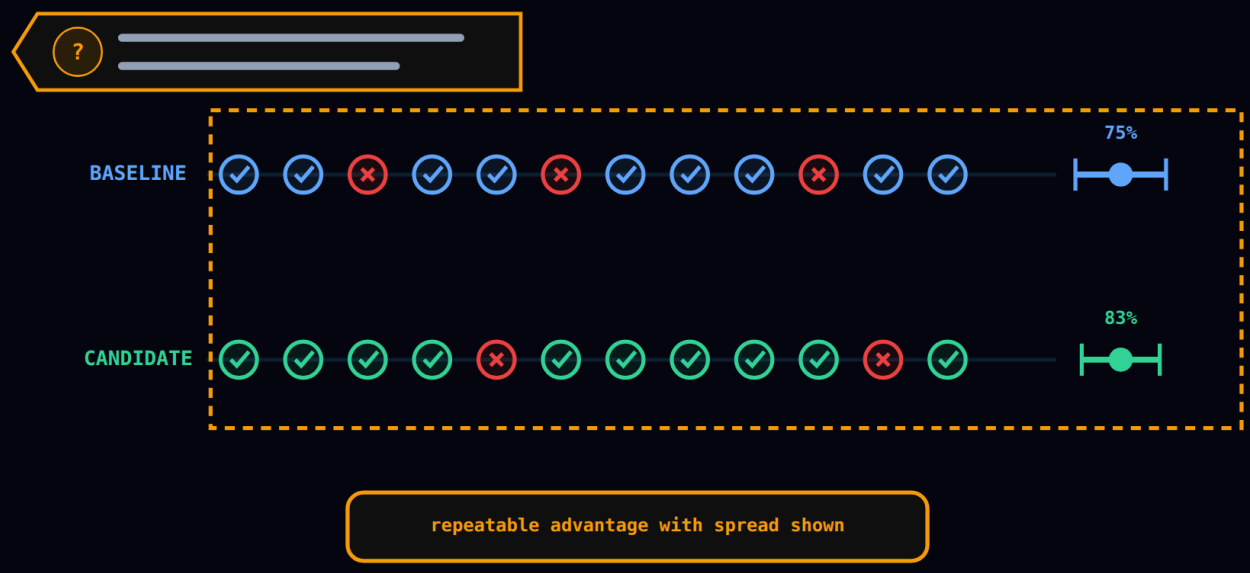
Three cases align across automation and people, but one confident approval conflicts with the human safety label. That disagreement enters review and becomes evidence for changing the rubric or judge prompt.

Use the cheapest grader that matches the decision, then measure agreement against people on risky cases. Next, repeated trials will show why even a well-calibrated grader should not decide from one sampled response.

STAGE 6 OF 8

Repeated Trials and Variance

★ If you remember one thing · Repeated trials reveal both the candidate improvement and the uncertainty hidden by one run.



06 REPEATED TRIALS AND VARIANCE

The same grader and case can produce different results when the model samples again. We will compare baseline and candidate outcomes on identical trials instead of trusting a single response.

One green candidate result looks encouraging, but the baseline also passes its first run. With one observation per system, neither average quality nor normal spread is available to compare.

Both systems passed once, yet later outputs may differ because sampling remains probabilistic. What will repeated trials reveal that these two green dots hide?

PAUSE AND PREDICT

What important evidence is missing after one trial?

- Only response speed
- Average and run-to-run spread
- The output schema

Skip this check

The rows expand to twelve aligned outcomes, and the candidate mean finishes higher while both systems still show failures. The spread markers make uncertainty part of the comparison instead of hiding it behind one score.

Try every Trials option and inspect the evidence readout as each row grows. The settings move from no spread estimate, through an uncertain direction, to a fuller comparison with variation shown.

Repeated trials do not remove randomness, but they make average quality and uncertainty measurable. Next, we will combine that evidence with live signals and decide whether a candidate should ship.

STAGE 7 OF 8

Production Feedback and Release

07 PRODUCTION FEEDBACK AND RELEASE

Repeated offline trials tell us about known cases, but production traffic contains new language, tools, and edge conditions. The evaluation system must listen after release instead of declaring the problem finished.

A thumbs down, blocked tool call, or human escalation is only a signal at first. Logging the input, system version, retrieved context, output, and outcome gives reviewers enough evidence to understand it.

Reviewers cluster useful failures into missing evidence, authorization, and policy ambiguity. Representative examples then become new versioned cases, which expands the offline suite with situations the team actually encountered.

The release gate now joins three independent facts: known cases pass, repeated trials show the candidate spread, and production guardrails remain healthy. If one pillar fails, the decision changes from ship to hold.

Reliable evaluation is a loop from product promises to tests, decisions, live evidence, and better tests. Now let us step back and connect the whole system before you build your own suite.

STAGE 8 OF 8

The Evidence System

08 THE EVIDENCE SYSTEM

We started with versioned golden cases, which hold realistic inputs and product promises steady while the system changes. They give every later measurement a reviewed foundation.

Then we protected exact machine contracts with program checks. Schemas, arguments, permissions, and side effects should fail clearly before uncertain behavior reaches production.

Next we separated retrieval coverage, claim grounding, and tool outcomes. That separation turns one weak answer score into a diagnosis of the component that needs repair.

We added model graders for scale and human labels for calibration. Their disagreement is useful evidence because it reveals judge bias, rubric gaps, or genuinely ambiguous product expectations.

We repeated the same comparison across multiple samples, which exposed both average quality and spread. This keeps a lucky output from carrying more confidence than the data supports.

Finally, production signals became reviewed regression cases. The offline suite now grows from real failures, so evaluation stays connected to changing users, policies, and system behavior.

The complete loop combines exact contracts, graded behavior, repeated evidence, and live feedback around one versioned dataset. That is how we test probabilistic software without pretending it behaves deterministically.