TOOLKIT · FREE RESOURCE

The AI Feature Evaluation Kit

A working kit for deciding whether an LLM feature is ready to ship. Define the job, build a golden set from real cases, classify failures, set cost and latency budgets, and apply release gates with a clear ship or no-ship rule.

By **Ratomir Jovanovic** · ratomir.com/resources/ai-feature-evaluation-kit

---

This kit is for product leaders, PMs and engineering leads who are shipping a feature built on a large language model and need a defensible answer to one question: is it good enough to put in front of customers, and how will we know if it gets worse? It assumes you already have a working prototype. It does not assume you have a data science team.

The outcome is a small, repeatable evaluation system that fits in one spreadsheet: a golden set of real cases, a shared failure taxonomy, a results summary that anyone can read, and release gates that turn "it looks fine in the demo" into a decision rule. The same system then protects you every time someone changes a prompt, swaps a model or adds a retrieval source.

How to use this

1. Write down the job the feature does and the failures you will not accept. Do this before you look at any outputs, so the outputs do not quietly lower your standards.
2. Open the **Golden Set** sheet in the Excel workbook and collect 50 to 150 real cases. Start with the 8 example rows to see the level of detail expected.
3. Agree the failure categories on the **Failure Taxonomy** sheet with the people who will review outputs, including at least one person who owns the downstream risk (support, legal, compliance or finance).
4. Run every case through the current version, have reviewers mark each output pass or fail, and tag every failure with one category and one severity.
5. Read the **Results Summary** sheet. It counts failures by category and severity and compares the run against the previous one.
6. Apply the **Release Gates** sheet. If any gate fails, the answer is no-ship, regardless of how good the averages look.
7. Re-run the full set on every prompt change, model change, retrieval change or parameter change. Treat the workbook as part of the release process, not a launch artifact.

The PDF guide walks through each step in more detail and includes reviewer instructions you can paste into your own process.

Step 1: Define the job and the unacceptable failures

Most LLM evaluations fail before the first test case is written, because nobody wrote down what the feature is for. "Helps agents reply faster" is a goal, not a job. A job is specific enough that two reviewers would agree whether a given output did it: "Produces a first-draft reply to an inbound payment dispute email that an agent can send after reading it once and editing no more than two sentences."

Next to the job, write the unacceptable failures. These are the outputs that would hurt a customer, create legal or financial exposure, or damage trust badly enough that one occurrence matters. They become your blocker category. Keep the list short and concrete. "Never promises a refund outcome the agent has not approved" is testable. "Never says anything inappropriate" is not.

Step 2: Build a golden set from real cases

A golden set is a fixed collection of inputs with a written description of what a good output must do and must not do. It is the single most valuable asset in this kit, and the quality of your decisions will never exceed its quality.

Build it from real cases, not from cases you imagined. Pull historical tickets, emails, chats or documents, remove personal data, and sample deliberately:

- The common path, roughly in proportion to real traffic, so averages mean something.
- The known hard cases: ambiguous requests, missing information, angry customers, multiple issues in one message.
- The risky cases: anything touching money, legal commitments, regulated topics, minors or account security.
- Adversarial inputs: instructions hidden in the user text, requests to ignore policy, attempts to extract other customers' data.
- Out-of-scope inputs the feature should decline or route elsewhere.

For each case, write the expected behavior and a "must not" in plain language. Do not write a single ideal answer. LLM outputs vary in wording, and grading against exact text will fail good outputs and teach the team to ignore the evaluation.

Fifty well-chosen cases will tell you more than five hundred random ones. Grow the set over time by adding every production failure you find, so the same mistake cannot return unnoticed.

Step 3: Use a failure taxonomy

A pass rate on its own hides what matters. A feature that passes 92% of cases with all failures being slightly long replies is very different from one that passes 95% with two failures that promise refunds. The taxonomy makes that difference visible.

Severity	Meaning	Release consequence
Blocker	Could cause financial, legal, security or serious trust harm on a single occurrence	Any blocker in the golden set means no-ship
Major	Wrong or unhelpful enough that a user would need to redo the work or escalate	Must stay under an agreed rate
Minor	Quality issue a user would fix quickly: tone, length, formatting	Tracked, does not block on its own

Keep the category list to six to eight items. More than that and reviewers start disagreeing about which bucket to use, which makes the counts unreliable. Every category needs a definition and one example, which is how the **Failure Taxonomy** sheet is laid out. When a new failure does not fit any category, add it to a notes column first and only create a new category if it repeats.

Step 4: Combine human and automated review

Human review is the reference standard. Automated checks are how you afford to run the evaluation often. You need both, and you need to know which one you trust for what.

Use **automated checks** for anything deterministic: required fields present, length limits, banned phrases, correct format, links that resolve, no account numbers in the output, correct language. These are cheap and should run on every change.

Use **model-graded checks** (one model scoring another's output against a rubric) for fuzzier criteria such as "addresses the customer's actual question". Before you rely on them, have humans grade the same 30 to 50 cases and compare. If the grader disagrees with humans often on a category, do not use it for that category's gate.

Use **human review** for blockers and for every release decision. Two reviewers per case on the first run, with disagreements discussed, will calibrate the team faster than any written rubric. After that, one reviewer plus a spot check is usually enough. Record reviewer names in the workbook so you can see when one person's standard drifts.

Step 5: Set cost and latency budgets

An output that is correct but arrives in twelve seconds, or costs more than the work it replaces, is still a failure. Set budgets before you optimise quality, because many quality improvements (longer prompts, bigger models, more retrieval, multiple calls) quietly spend them.

Express both as percentiles, not averages. A median latency of two seconds with a 95th percentile of fifteen seconds means one in twenty users waits fifteen seconds. Express cost per completed task, including retries and any grading or guardrail calls, not per API request. Then write the budget next to the value it must stay below: for example, the cost of the agent time the draft saves.

Step 6: Run regression checks on every change

The most common way an LLM feature degrades is not a model failure. It is a well-intentioned prompt edit that fixes one complaint and breaks three cases nobody re-tested. Treat prompts, model versions, temperature, retrieval sources and tool definitions as code: version them, and re-run the full golden set before any change reaches production.

The **Results Summary** sheet compares the current run with the last accepted run category by category. Look for any case that passed before and fails now, even if the overall rate went up. A new blocker in a previously passing case is a regression, and a regression in a blocker category is no-ship.

When a provider updates a model behind the same name, run the set as if you had changed it yourself. Schedule a periodic re-run even when nothing on your side changed.

Step 7: Apply release gates and a ship or no-ship rule

Release gates are thresholds agreed before the run. Agreeing them afterwards invites the team to move the line to wherever the results landed. The **Release Gates** sheet holds them, with a reason for each so future readers know why a number was chosen.

The rule is simple: **ship only if every gate passes. If any gate fails, do not ship, and record which gate failed and what will change before the next run.** No averaging across gates, no "two out of three is fine". If a gate is wrong, change it in writing, with the reason, before the next run, not during the release meeting.

A typical starting set of gates:

Metric	Threshold	Why
Blocker failures in golden set	0	One occurrence can cause real harm
Overall pass rate	at least 90%	Users stop trusting a draft tool that is wrong often
Major failures	at most 5% of cases	Keeps rework below the time saved
Regressions versus last release	0 cases moving from pass to blocker	Changes must not reintroduce fixed risks
p95 latency	within budget	Slow tails drive abandonment
Cost per completed task	within budget	The feature must cost less than the work it replaces

Worked example: drafting replies to payment dispute emails

This is a hypothetical example built to illustrate the method. It does not describe a real company or real results.

A mid-sized payments company wants an assistant that drafts replies to customers who email about disputed card transactions. Agents review every draft before sending. The job statement: "Draft a reply an agent can send after one read and at most two sentence edits, which acknowledges the dispute, states the correct next step under our policy, and requests only the information we actually need."

The team lists its unacceptable failures: promising a refund or outcome, giving a timeline that contradicts the card scheme rules in the policy document, asking the customer to send full card numbers or security codes, and following instructions embedded in the customer's email. These become blockers.

They build a golden set of 80 anonymised emails from the last six months. The workbook includes 8 of them as examples, covering a straightforward unrecognised charge, a merchant that did not deliver, a duplicate charge, a customer asking for an outcome guarantee, a dispute past the filing window, an email in another language, an email containing a hidden instruction, and a furious customer threatening to go to a regulator.

On the first run, human reviewers find 71 of 80 passes (89%). Of the 9 failures, 6 are minor tone and length issues, 2 are major (the draft asked for documents the policy does not require), and 1 is a blocker: on the "past the filing window" case the draft told the customer the bank would "look into getting your money back", which reads as a promise. The pass rate is close to the gate, but the single blocker makes the decision no-ship regardless.

The team adds an explicit policy rule to the prompt and a retrieval step for filing windows, then re-runs the full set. The blocker clears, the pass rate rises, but one previously passing case (the duplicate charge) now fails as major because the longer prompt pushed the model to over-explain scheme rules. That is exactly what regression checking is for. After a further prompt edit, the third run passes every gate, p95 latency is inside budget, and the feature ships to a group of agents with the production failure log feeding new cases back into the golden set.

Common mistakes

- **Grading against a single ideal answer.** You will fail good outputs and lose the team's trust in the evaluation. Grade against behaviors.
- **Building the golden set from imagined cases.** Imagined cases are easier than real ones. Use real traffic, then add the hard and adversarial cases deliberately.
- **Letting the averages hide blockers.** A 96% pass rate with one blocker is a no-ship. Read the failures, not just the rate.
- **Trusting a model grader you have not checked.** Compare it with human grades first, category by category.
- **Setting gates after seeing the results.** Gates agreed afterwards are rationalisations.
- **Evaluating once at launch.** Most damage happens in later prompt edits and silent model updates.
- **Leaving risk owners out of the taxonomy.** If support, legal or finance did not help define blockers, you will discover their definitions after launch.

When not to use this kit

If the feature is still being explored and you do not yet know what job it does, a formal golden set is premature. Spend the time on user conversations and rough prototypes instead, then come back here once the job is clear. The kit is also more than you need for low-stakes internal tools where a wrong output costs a few seconds of someone's time. And it is not a substitute for a regulatory, security or model risk review where those apply. It gives you the evidence for those reviews, not their approval.

Editable template: Evaluation workbook (Excel), available at ratomir.com/resources/ai-feature-evaluation-kit

© 2026 Ratomir Jovanovic. Free to use inside your team. Please link to ratomir.com/resources/ai-feature-evaluation-kit when you share it.