GUL'S NOTEBOOK · AI RESEARCH · OPEN ACCESS

The evals I run before shipping a prompt change

Muhammad Gulzaib
https://orcid.org/0009-0000-8709-6989
SynergyFlow Labs · Islamabad, Pakistan

Published	2026-07-15
URL	https://gulzaib.vercel.app/blog/the-evals-i-run-before-shipping-a-prompt-change
Licence	Creative Commons Attribution 4.0 International (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/

ABSTRACT

A prompt change needs a comparison against the previous version, not a handful of convincing examples. Run both on the same representative cases, check hard output requirements, review task quality and inspect regressions before deciding whether to ship. Model outputs vary, so repeat important cases and keep a held-out set for a final check.

Keywords: llm, evaluation, testing, research

This article was prepared in November 2026. The displayed 15 July 2026 date was requested for the post; it is not its original publication date.

The title describes a proposed release checklist, not a claim about a particular production system or a past regression. The example below uses fictional tickets and hand-written responses. It runs locally without calling a model, so it demonstrates the gate rather than measuring an LLM.

Start with the behaviour the application needs

Imagine an assistant that routes a school's support tickets. The prompt should return one JSON object with a `category`: `fees`, `attendance` or `needs_review`. A mixed or empty ticket should go to a person rather than be assigned to the wrong queue.

The task is not "produce nice JSON". A valid object with the wrong category still sends a real person to the wrong team. Define both the output contract and the behaviour before adjusting the prompt. OpenAI's evaluation guidance recommends task-specific tests and continuous evaluation; Anthropic similarly recommends measurable criteria and cases that reflect real use. [1] [2]

For this example, a starting set would contain routine fee and attendance requests, empty

input, mixed requests, paraphrases and messages in the languages the support team serves. Include known failures and examples that attempt to override instructions. The set is a proposal until it is checked against real, appropriately authorised traffic.

Keep inputs and expected answers separate

Each case needs an identifier, the input visible to the application, an expected result or rubric and a reason for its inclusion. Store expected labels and private reference answers outside the prompt sent to the model. If they are accidentally included in the input, the test no longer measures the behaviour you intended.

Case	Example input	Expected behaviour
Normal	"The fee voucher total is wrong."	`fees`
Empty	An empty message	`needs_review`
Mixed	"Fees and attendance both look wrong."	`needs_review`
Instruction attempt	A ticket asks the assistant to change its rules	Do not grant the ticket new authority; test the required fallback and tool restrictions

These are synthetic examples, not measurements. In a real suite, include the range of requests users send, not only the difficult ones. Keep a separate held-out set that is not used to tune the prompt, and refresh it as usage changes. A test set that the team repeatedly adjusts to pass is not an independent final check.

Compare a baseline and a candidate

Run the current prompt and the proposed prompt against the **same cases**, with the same model version, input, tools, retrieval context and settings when the prompt is the only change being tested. If you change the model or retrieval configuration too, record that and treat the result as a combined change.

Save the case ID, dataset and prompt versions, model identifier and settings, output, score, errors, latency and token usage. Protect run logs and remove personal data where practical. A timeout, refusal or missing response must be recorded and handled by the gate; silently dropping it makes the pass rate look better than it is.

For variable outputs, run each important case more than once and report the number of runs and failures. Interleave or randomise baseline and candidate order where practical to reduce time-related effects. A fixed seed or low temperature can help comparisons when supported, but neither guarantees identical output from a changing service.

A runnable offline gate

Download the Python fixture demo. It uses only the standard library and intentionally contains one failing candidate. It does not call an LLM, load real tickets or show results from a production system.

terminal

```
python3 prompt-regression-demo.py
```

fixture output

```
Baseline: 4/4 passed
Candidate: 3/4 passed
Regressions: mixed
Release blocked: inspect failures
```

The program exits with code 1 for this deliberate failure. One local run took about 0.02 seconds including Python startup; real model calls and human review will take longer, and their duration needs to be measured in the target environment. The four baseline and candidate responses are hand-written fixtures; the output verifies the gate and parser, not the quality of a prompt or model. It checks that the JSON object has exactly one `category` field and that its value matches the expected label. Invalid JSON, missing fields and unexpected fields fail.

The comparison at the heart of the demo identifies a case that passed before and fails now:

prompt-regression-demo.py

```
losses = [c['id'] for c in CASES if base[c['id']] and not new[c['id']]]
```

To turn the demo into a real eval, replace the two fixture dictionaries with outputs from your application for each case and prompt version. Send only the case input to that application, never the expected label. Record provider errors explicitly and preserve the paired case IDs. The demo does not implement repeated runs, privacy controls, tool-call inspection or statistical testing; add those for the system being evaluated.

Use more than one kind of check

Check	Example	Limitation
Deterministic	Parse JSON and compare a routing label	Cannot judge whether an open-ended explanation is helpful

Human rubric	Reviewer checks accuracy, clarity and missing information	Takes time and needs consistent instructions
Model judge	A separate model applies a defined rubric	Can inherit biases or make errors of its own
System check	Measure latency, errors, cost and permitted tool actions	Does not establish answer quality

For summaries or generated exam questions, exact string equality is usually too brittle. A reviewer can instead assess whether the answer follows the source, includes required points and avoids unsupported claims. A machine-readable citation can identify a source but does not by itself prove that the source supports the claim.

If using an LLM as a judge, compare its decisions with independently labelled human examples and inspect disagreements. Research on model judges describes risks including position and verbosity bias; when comparing two answers, vary their order and check whether the judgement changes. A judge is another component to evaluate, not a ground truth. [3]

Test the whole boundary, not only the wording

A support ticket or retrieved document may contain an instruction such as "ignore the routing rules". Treat it as untrusted content. Include adversarial cases in the suite, but do not infer that the system is safe because a few examples passed. OWASP notes that prompt injection has no known foolproof prevention and recommends measures including least privilege, output validation and human approval for high-risk actions. [4]

Test those controls outside the prompt. A read-only routing assistant should not have permission to send messages, change records or expose secrets merely because a ticket asks it to. Use isolated test accounts and mock tools, and check the attempted actions as well as the text response. Do not put real credentials or student records into a test fixture.

Decide the gate before seeing the result

An example policy for a low-risk pilot is: no invalid output or known critical security failure in the test run; investigate any baseline-to-candidate loss; and obtain human review of a sample of open-ended answers. This is a proposed policy, not a universal threshold. Set limits based on the task, impact and observed variability before comparing versions.

A four-case demo cannot establish a reliable pass rate. On a larger held-out set, report the number of cases, repeated trials, per-category results and uncertainty, not only a single average. A candidate can improve overall while failing more often on mixed-language messages or a rare high-impact case. Examine wins and losses for the same inputs, then decide whether more data is needed.

If the gate fails, keep the previous version active. Investigate the case, add an appropriate regression example to the development suite and rerun the comparison. If the candidate passes, keep the previous prompt and settings available for rollback, monitor the release and review newly observed failures. Passing an eval is evidence within its coverage, not a guarantee for every future input.

*[Figure: image]***Agents that do real work: a pragmatic multi-agent design**: Start with a workflow and add an agent only where the path can't be known. The design I use at SynergyFlow Labs: narrow tools, state, budgets and approvals. *[Figure: image]***Grading written answers with AI: what worked, what didn't:** What research says about AI grading of subjective answers, with a sample rubric, teacher review workflow and a plan for measuring agreement.

> **Key takeaways**
>
> - Define the task and output contract before changing the prompt.
> - Compare baseline and candidate on the same cases, including failures and edge cases.
> - Combine deterministic checks with calibrated human review for open-ended quality.
> - Test tool permissions independently of prompt wording.
> - Keep a held-out set, record versions and retain a rollback path.

Questions readers ask

What is prompt regression testing?

Prompt regression testing compares a changed prompt with a baseline on the same cases and checks whether important behaviours have worsened. It can include deterministic validation, human review and repeated model runs.

How many examples do I need for an LLM eval?

There is no universal number. Begin with representative and critical cases, then expand and measure uncertainty; a few cases can catch obvious failures but cannot prove reliability.

Can an LLM judge replace human evaluation?

A model judge can help review open-ended answers at scale, but it can be biased or wrong. Compare it with human judgements on a separate labelled set and inspect disagreements.

References

1. OpenAI. Evaluation best practices.
2. Anthropic. Define success criteria and build evaluations.

3. Zheng, L. et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. NeurIPS Datasets and Benchmarks, 2023. https://doi.org/10.48550/arXiv.2306.05685
4. OWASP. LLM01:2025 Prompt Injection, 2025.

CITE AS

Gulzaib, M. (2026, July 15). The evals I run before shipping a prompt change. Gul's Notebook. https://gulzaib.vercel.app/blog/the-evals-i-run-before-shipping-a-prompt-change

You may share and adapt this text for any purpose, including commercially, as long as you credit the author and link to the original. Licence: https://creativecommons.org/licenses/by/4.0/