Make an AI experiment repeatable.

Record the task, inputs, candidate configuration and failures so a result can be compared rather than remembered.

Evaluation record

Check	Evidence to retain
Dataset version	Representative tasks, difficult examples and expected refusals
Candidate configuration	Model identifier, prompt, retrieval settings and tool permissions
Outcome	Task result, source support, latency and cost
Release decision	Accepted limitations, owner and rollback condition

Keep development and evaluation examples distinct

Repeatedly tuning against the same small set can produce a system that passes the notebook but fails ordinary work. Retain a separate set for the release decision.

Inspect failures by category

Separate unsupported answers, permission failures, tool errors and missing context. Different failure types need different changes.

Start with the release decision

Identify what the evaluation will decide: whether to change a model, alter retrieval, add a tool or expand access to more users. Record the current behaviour and the improvement expected. Without a baseline, a collection of attractive answers can look convincing while providing little evidence of progress.

Collect representative tasks, including ordinary requests, incomplete records, ambiguous questions and cases that should be refused or escalated. Keep sensitive examples under appropriate access controls. Give each case a stable identifier so the same task can be compared across changes. Explain how examples were selected and which users or workflows they do not represent. The evaluation set should reflect the intended use rather than only the examples that are easy to pass.

Make results repeatable and reviewable

Record the application version, model identifier, prompt version, retrieval settings and source collection for each run. Save expected behaviour separately from the generated response. When a result is incorrect, explain why. An irrelevant citation, an incorrectly extracted total and an unauthorised tool call are different failures that need different remedies.

Use automated checks where the expected result is well defined, and informed human review where judgement is necessary. Record disagreements between reviewers rather than hiding them inside an average score. Keep release evaluation cases separate from examples repeatedly used during development. Otherwise, the team may tune the system to familiar questions without improving performance on the work users actually bring.

Connect findings to operating limits

Review failures by business consequence. A wording preference should not carry the same weight as exposing a restricted document or proposing a duplicate payment. Define the conditions that block release, and record an owner for any accepted limitation. Compare response time and operating cost with quality where they affect the usefulness of the workflow.

Finish with a decision and the scope it supports. A system may be ready for an internal pilot where every output is reviewed, while remaining unsuitable for unattended use. Record that boundary explicitly. Repeat relevant evaluations after source changes, model upgrades and meaningful incidents. The notebook becomes a maintained explanation of why the system is trusted for a particular task.

Should we average all results into one score?

Keep critical controls separate. A permission breach or unauthorised action should be visible even when most answers are useful.

Read the current resource online

https://cobnex-review.rgcsekaraa.workers.dev/resources/evaluation-notebook/