Lesson 5 · RAG Evaluation & Regression Testing

PDF: this lesson · **Install (Linux · macOS · Windows):** guide

> **Part of local-ai-lab** - a hands-on course for building local AI.

> **Interactive version (slides):** https://nikolareljin.github.io/local-ai-lab/lesson-5-rag-evaluation-regression-testing.html **Read it locally (no GitHub Pages):** `./run -l 5 lesson` **Course home:** https://nikolareljin.github.io/local-ai-lab/ **Source:** https://github.com/nikolareljin/local-ai-lab **Author:** Nik Reljin **Time:** ~30-45 min · **Prerequisites:** Lesson 1 (Lessons 3-4 helpful) · full objectives in SYLLABUS.md

> **Lessons:** 1 · RAG → 2 · MCP → 3 · Hybrid retrieval → 4 · RAG safety → **5 · RAG evaluation (you are here)** → 6 · Repo assistant → 7 · LangChain → ... → 15 · Docs from changes

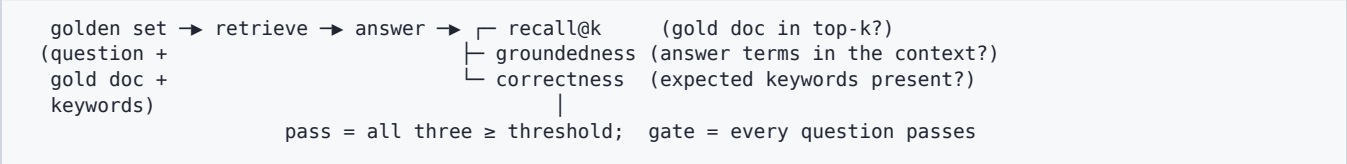
> **Status: working demo.** Runnable in **Python, Node.js, and C# / .NET** - same algorithm, three languages, identical output. Runs 100% offline, no model required. See **From demo to production** at the end for what to harden for real use.

What you'll learn

Lessons 1 and 3 made retrieval **work** and made it **better**; Lesson 4 made it **safe**. This lesson makes "better" and "safe" **measurable**. The golden rule of RAG evaluation:

> **If you can't put a number on it, you can't tell when it breaks.**

You score a pipeline against a small **golden set** and gate on the result, so a regression shows up as a failed check instead of a customer complaint.



By the end you'll understand:

- **Golden questions** - pairing a question with the document that **should** be retrieved and the facts a correct answer must contain
- **Retrieval recall@k** - did the gold document come back in the top-k results?
- **Groundedness** - is every claim in the answer supported by the retrieved context?
- **Answer correctness** - does the answer actually contain the expected facts?
- **Gating & regression testing** - turning the three scores into one pass/fail you can put in CI

> **The one idea:** quality you can't measure is quality you can't defend. Three numbers and a gate turn "seems good" into "here's the score, and here's the line it must stay above."

The demo

A tiny corpus of five support docs lives in `data/`, and the golden set that scores them is `data/golden.json` - five questions, each tagged with its **gold document**, the **answer keywords** a correct answer must contain, and the **thresholds** the gate enforces.

We run the golden set under **two configs** and print both scorecards plus a regression summary:

- **baseline** - `top_k=3`, no padding: clears the gate
- **candidate** - `top_k=1`, an answer padded with one unsupported sentence: a reasonable-looking tweak that quietly regresses two numbers

Run the comparison (offline, no dependencies)

From the repo root - the `demo` action prints the result and exits, with **nothing to install** (pure standard library). Pick any language; all three give the **same** output:

```
./run -l 5 demo                # Python - print both scorecards + the regression summary and exit
./run -l 5 --lang node demo    # Node.js - same output
./run -l 5 --lang csharp demo  # C# / .NET 8 - same output
./run -l 5 test                # the offline Python test
./run -l 5 lesson              # read this lesson in a browser, served locally
./run -l 5 show                # walk through this lesson's steps (code, data, prompts, commands)
```

Output:

```
Config: baseline  (top_k=3, padding=off)
 id               recall  grounded  correct  result
 refund-window     1.00      1.00     1.00  PASS
 shipping-speed    1.00      1.00     1.00  PASS
 cannot-login      1.00      1.00     1.00  PASS
 warranty-length   1.00      1.00     1.00  PASS
 reset-password    1.00      1.00     1.00  PASS
 Aggregate: recall 1.00  grounded 1.00  correct 1.00   5/5 passed   GATE: PASS

Config: candidate  (top_k=1, padding=on)
 id               recall  grounded  correct  result
 refund-window     1.00      0.63     1.00  FAIL
 shipping-speed    1.00      0.61     1.00  FAIL
 cannot-login      1.00      0.56     1.00  FAIL
 warranty-length   1.00      0.68     1.00  FAIL
 reset-password    0.00      0.56     1.00  FAIL
 Aggregate: recall 0.80  grounded 0.61  correct 1.00   0/5 passed   GATE: FAIL

Regression vs baseline:
 mean recall:       1.00 -> 0.80   (-0.20)
 mean groundedness: 1.00 -> 0.61   (-0.39)   below threshold 0.75
 mean correctness:  1.00 -> 1.00   (+0.00)
 gate:              PASS -> FAIL
```

The candidate **looks** fine - the answers still read well and still contain the right keywords (correctness stays at 1.00). But recall fell because a smaller top-k dropped one question's gold document, and groundedness fell because the padded sentence is unsupported. That's exactly

the kind of regression a manual eyeball check waves through and a gate catches.

Experiment in the scorecard (needs Flask)

For a hands-on feel, bare `./run -l 5` opens an interactive **scorecard** over the same golden set. Leave the box empty for the whole-set card (the three means + the gate), or pick a question to drill into its numbers - then move the sliders or flip on unsupported padding and watch the gate flip.

```
./run -l 5           # opens http://127.0.0.1:<port> - the scorecard playground over the golden set
```

Control	What moves
`top_k` → 1	`reset-password` recall drops to 0 (its gold doc ranked second)
unsupported padding on	groundedness drops below its gate for every question; correctness holds
groundedness / correctness gate sliders	change where the pass/fail line sits

> The playground is a small Flask app, so unlike the `demo` it needs one dependency. `./run` installs it into the project venv automatically on first use.

> **Why is the answerer deterministic?** To keep the lesson offline and reproducible, the answerer is a tiny extractive stand-in: it returns the fact from the top retrieved document (optionally padding in one unsupported sentence). Swap in your real pipeline and the golden set and metrics are unchanged.

> **Why JSON + deterministic scoring (not YAML + an LLM judge)?** This lesson deliberately uses a JSON golden set and three heuristic metrics so the whole thing runs **offline, model-free, and byte-identically across Python, Node.js and C#** - the same constraint every lesson in this course honours, and what makes the regression **reproducible** rather than a coin-flip. An LLM-as-judge, groundedness against cited spans, and out-of-corpus "not found" questions are real and valuable - they are the natural next step once a model is in the loop, covered under **From demo to production** below. The point here is the **scaffold**: a versioned golden set scored on tracked numbers behind a gate. Keep that shape and you can upgrade the scorer (JSON→YAML, keywords→judge) without changing the lesson.

Concept 1 · Golden questions - the source of truth

You can't measure a pipeline without knowing the right answer. A **golden set** encodes that: each question is paired with the document(s) that **should** be retrieved and the keywords a correct answer must contain.

```
{
 "id": "reset-password",
```

```
  "question": "how do i reset my password",
  "gold_docs": ["password_reset.md"],
  "answer_keywords": ["reset", "password", "sign"]
}
```

Curate it like code - version it, review changes, and **add a case for every regression you ever hit**, so the eval only ever gets stricter.

Concept 2 · Recall@k - did retrieval find it?

The first thing that can go wrong is retrieval simply not returning the relevant document. **Recall@k** is the fraction of a question's gold documents that appear in the top-k results - `1.0` when the gold document comes back, `0.0` when it falls out. It is blind to **how** retrieval works, so it applies to BM25, embeddings, or the hybrid from Lesson 3 equally.

Concept 3 · Groundedness - is the answer supported?

A fluent answer can still invent facts. **Groundedness** is the fraction of the answer's meaningful terms that actually appear in the retrieved context:

```
groundedness = len(answer_terms & context_terms) / len(answer_terms)
```

The demo's candidate pads every answer with **"A complimentary gift card will be mailed separately."** - no document says any such thing, so those terms are not in the context and groundedness drops. This is the metric that catches a confident hallucination.

Concept 4 · Correctness - is the answer right?

Grounded is not the same as right. **Correctness** is the fraction of the expected keywords present in the answer - a cheap, deterministic stand-in for "does it contain the facts we wanted?" In the demo it **stays at 1.00** under the candidate, which is the whole point: a regression can leave the obvious signal untouched while quietly breaking the others.

When each metric helps

Metric	Catches	Blind to	Cost
Recall@k	the relevant document never being retrieved	answer quality once the doc is present	negligible
Groundedness	fluent answers that invent unsupported claims	a grounded answer that is simply wrong	negligible
Correctness	answers missing the expected facts	**how** the answer was produced; hallucinated extras	negligible

No single number is enough - which is why the **gate** requires all three, and why you grow the golden set until it pins the failures you actually care about.

Polyglot by design

The retriever, the answerer stand-in and the three metrics are language-agnostic, so this lesson ships in **Python, Node.js, and C# / .NET** - each dependency-free, each reading the same `data/golden.json` and `data/` corpus, each producing byte-identical output.

Port	Entry point	Run
Python	`python/eval_rag.py`	`./run -l 5 demo` · `./run -l 5 test`
Node.js	`node/eval_rag.mjs`	`./run -l 5 --lang node demo`
.NET 8	`dotnet/Program.cs`	`./run -l 5 --lang csharp demo`

> All three commands are declared once in `lesson.json` - the single source of truth the `./run` engine reads. `./run -l 5 show` renders this lesson's elements in order. The interactive scorecard (`./run -l 5`) is Python-only, by convention shared across the course.

Exercises

- **Grow the set:** add a golden question whose gold document ranks third, then find the smallest `top_k` that still passes the whole set.
- **Tighten the gate:** raise the groundedness threshold to `0.9` - which questions fail, and is the baseline answerer actually that grounded?
- **A correctness trap:** write a question whose keywords a **wrong** document also contains. Which metric still catches the error - recall, groundedness, or neither?
- **Better metrics:** replace keyword correctness with embedding similarity or an LLM-as-judge, and confirm the baseline still clears the gate.

From demo to production

- **Keep the golden set, swap the pipeline** - point `evaluate` at your real retriever and model; the metrics and the gate are unchanged.
- **Grow the set from incidents** - every bug becomes a new golden question, so coverage only increases.
- **Strengthen the metrics** - pair keyword correctness with an LLM-as-judge or embedding similarity, score groundedness against cited spans, and track **latency and cost** alongside quality.
- **Test "not found" too** - add out-of-corpus golden questions whose correct answer is **"I don't know"**, and score whether the system abstains instead of confabulating. Honest refusal is a quality metric, not a failure.

- **Gate in CI** - fail the build when a tracked number drops below threshold, and store each scorecard so you can watch trends over time.
- **Fold in safety** - add Lesson 4's poisoned-document cases so a safety regression also shows up as a number.

Next lesson

Lesson 6 · Repo-aware AI assistant - ground an assistant in your own codebase so it answers with repo-specific context instead of generic guesses.

Course: nikolareljin.github.io/local-ai-lab · **Author:** Nik Reljin