

✦ GUIDE · RAG SYSTEMS

RAG Beyond The Demo

Why RAG assistants that shine in demos stumble in production — messy documents, tables, stale content, ambiguous questions, multiple languages — and the specific fixes for each failure mode.

FORMAT	LEVEL	READ TIME	UPDATED
Guide	Intermediate	20 min	May 2026

OVERVIEW

What this guide covers

A RAG demo is usually built on a handful of clean documents and questions written by the people who built it. Production brings scanned PDFs, versioned policies, spreadsheets, vague questions, mixed languages and users who notice every wrong answer. This guide is organized by failure mode: how each one shows up, how to diagnose it from traces, and what to change in ingestion, retrieval or generation to fix it — plus the evaluation and feedback loops that stop it coming back.

KEY TAKEAWAYS

- Most production failures are retrieval and ingestion problems that look like model problems.
- Diagnose from traces: first check whether the right passage was retrieved, then whether it was used.
- Tables, scanned pages and versioned documents need dedicated handling, not generic chunking.
- Ambiguous questions should get a clarifying question, not a confident guess.
- Freshness and deletion are features — define how quickly changes must appear in answers.
- Production feedback must flow into the eval set, or the same failures recur.

WHO IT'S FOR

Teams with a RAG pilot heading to production

AI / ML engineers debugging answer quality

Product owners of knowledge assistants

Support and operations leads using RAG tools

HOW TO USE IT

Start with the triage method in section 01. When you see a failure in production, find its symptom in sections 02–05 and apply the fixes. Section 06 sets up the loop that keeps quality from drifting.

CONTENTS

01 Why demos lie, and how to triage
02 Messy documents and tables
03 Stale, versioned and conflicting content
04 Ambiguous and multilingual questions
05 Generation failure modes
06 Evaluation and feedback loops

01 / DIAGNOSIS

Why demos lie, and how to triage

Demos pass because the documents are clean and the questions are friendly. In production, every wrong answer has a cause somewhere in the pipeline — find which stage before changing anything.

THE DEMO	PRODUCTION
A few dozen clean, born-digital documents	Thousands of files: scans, slides, spreadsheets, duplicates
Questions written by the build team	Short, vague, misspelled questions from real users
One language, one version of each policy	Mixed languages, outdated and conflicting versions
Success judged by reading a few answers	Every wrong answer noticed, screenshotted and shared

Triage a bad answer in this order

01 **Is the answer in the corpus?**
If not, the right behavior is a clear “I don’t know” — check that it refused.

02 **Was the source parsed correctly?**
Open the stored chunk text. Garbled OCR, lost tables or missing sections are ingestion bugs.

03 **Was it retrieved?**
Check whether the right chunk appeared in the candidates before and after reranking.

04 **Did it fit in context?**
Check whether it survived context assembly or was cut by token limits.

05 **Did the model use it correctly?**
Only now look at the prompt, the model and the answer format.

✦ 02 / DOCUMENTS

Messy documents and tables

Parsing quality sets the ceiling for everything downstream. If the text in the index is wrong, no retriever or model can recover it.

SYMPTOM	LIKELY CAUSE	FIX
Answers mix up columns or rows	Tables flattened into running text	Extract tables as structured data; store as Markdown or row groups with headers repeated
Numbers wrong in answers from scans	Low-quality OCR	Use layout-aware OCR; store confidence; route low-confidence pages to review
Answers quote headers, footers or disclaimers	Boilerplate indexed as content	Strip repeated page furniture before chunking
Slide decks retrieve poorly	Sparse text without context	Combine slide title, body and speaker notes; add deck title as context
Same passage appears many times	Duplicate files and copies across folders	Content hashing and de-duplication at ingestion, keeping the canonical source

01

Inspect the index, not the source

Build a simple tool to view the exact chunk text, metadata and neighbors stored for any document.

02

Route by document type

Use different parsers for born-digital PDFs, scans, spreadsheets, slides and HTML rather than one generic path.

03

Spreadsheets are data

For numeric questions over tabular data, query the table (SQL or a tool) instead of embedding cells as text.

04

Keep a parsing test set

A small set of tricky files with expected extracted text catches parser regressions.

03 / FRESHNESS

Stale, versioned and conflicting content

Users trust an assistant until it cites last year's policy. Freshness needs a target, a pipeline and a way to resolve conflicts.

01 **Set a freshness target**

Agree how quickly a change or deletion in the source must appear in answers, per source, and monitor it.

02 **Ingest from change feeds**

Use incremental sync with updates and deletions, not periodic full re-crawls that miss removals.

03 **Model versions explicitly**

Store effective dates and version status; filter superseded versions out of retrieval by default.

04 **Prefer authority on conflict**

Rank by source authority and recency metadata; when sources disagree, say so and cite both.

05 **Expire what nobody owns**

Flag documents with no owner or long-untouched content for review rather than serving them silently.

FIELD NOTE

Deletion is the freshness bug nobody tests

Add a test that deletes a document in the source system and asserts it stops appearing in retrieval within the agreed window. Many pipelines handle updates correctly and never remove anything.

✦ 04 / QUERIES

Ambiguous and multilingual questions

Real questions are short, vague and written in whatever language the user thinks in. Retrieval needs help understanding them.

Query failure modes

SYMPTOM	EXAMPLE	FIX
Too vague to retrieve well	“leave policy”	Ask a clarifying question when top results span distinct topics or audiences
Depends on conversation	“what about for contractors?”	Rewrite follow-ups into standalone queries using chat history
Missing user context	“how many days do I get?”	Add role, location and team from the user profile as filters
Jargon and acronyms	Internal product codes or abbreviations	Maintain a glossary for query expansion; rely on keyword search for exact codes
Compound question	Two questions in one message	Decompose, retrieve for each part, then answer together

Multilingual retrieval

EMBED

Use multilingual embeddings

Choose an embedding model evaluated on your languages so a question in one language can find passages in another.

INDEX

Store language as metadata

Detect language per chunk to filter, boost or route, and to spot gaps in coverage.

KEYWORD

Language-aware keyword search

Configure tokenizers and analyzers per language; default analyzers often fail on non-Latin scripts.

ANSWER

Answer in the user’s language

Retrieve across languages, cite the original source and answer in the language of the question.

Add questions in every supported language — including code-mixed questions that combine two languages in one sentence — to the evaluation set. Quality in the build team’s language says little about the others.

✦ 05 / ANSWERS

Generation failure modes

When retrieval is right and the answer is still wrong, the problem is usually context assembly or answer policy, not the model.

SYMPTOM	LIKELY CAUSE	FIX
Confident answer with no real support	No relevance threshold	Decline or ask when reranker scores fall below a calibrated threshold
Correct source cited, wrong detail	Answer drawn from a nearby chunk	Include neighboring chunks or the full section around the best match
Answer ignores a key exception	Exception in a separate chunk	Retrieve sibling sections; keep clauses and their exceptions together at chunking
Overlong, hedged answers	Prompt asks for completeness over precision	Specify format, length and when to quote verbatim
Citations that do not match claims	Citations added after the fact	Require per-claim citations to retrieved chunk IDs and check them automatically

✦ 06 / LOOP

Evaluation and feedback loops

Production is the best source of test cases you will ever have. Wire it back into the eval set so each fixed failure stays fixed.

01 User flags or low score → 02 Trace triaged to a stage → 03 Fix in ingestion / retrieval / prompt → 04 Case added to eval set → 05 Eval gates the release

SIGNALS

- ☐ Thumbs up / down with an optional reason on every answer
- ☐ Implicit signals: rephrased questions, copied answers, escalations

- [ ] Queries with no results or low retrieval scores logged

EVALUATION

- [ ] Eval set built from real production questions, grouped by failure mode
- [ ] Retrieval and answer metrics tracked separately
- [ ] Parsing, freshness and multilingual cases included

PROCESS

- [ ] Weekly review of flagged answers with a domain expert
- [ ] Every fix links to the trace and the new eval case
- [ ] Quality dashboard per source, language and query type

> **The demo proves it can work. Production proves it keeps working — and the eval set is how you know.**
>
> SPLENDENSLABS ENGINEERING PRINCIPLE

✦ GET IN TOUCH

Let's Talk

We take RAG pilots from impressive demo to dependable production system. Bring us your failing questions and your traces, and we will find where the pipeline breaks.

- A RAG quality audit on your traces
- Ingestion and parsing fixes for messy content
- Evaluation and feedback loop setup
- A production hardening plan

BOOK AN AI STRATEGY SESSION →

EMAIL
hello@splendenslabs.com

WEB
splendenslabs.com

OFFICE
Sona Incubation Foundation, Salem