

# Retrieval-Augmented Generation for Internal Knowledge Bases

## Architecture, Failure Modes, and an Evaluation Checklist

*A technical white paper*

---

### Executive Summary

Most organizations already own the answers their employees are looking for. The trouble is that those answers are scattered across wikis, ticket histories, policy documents, shared drives and chat archives, and finding them takes longer than it should. Large language models promise a conversational front door to all of that material, but a model on its own knows nothing about a company's private documents, and when it does not know, it can produce a fluent, confident and wrong answer.

Retrieval-augmented generation (RAG) addresses both problems with a simple idea: before the model answers, the system searches the organization's own content, hands the most relevant passages to the model, and instructs it to answer from those passages. The model supplies language; the knowledge base supplies facts.

The idea is simple. Getting it right is not. A RAG system is a chain of components — ingestion, chunking, embedding, retrieval, ranking, prompting and generation — and a weakness in any link shows up to the user as a bad answer. Teams that treat RAG as "a model plus a vector database" tend to ship a demo that impresses and a product that disappoints.

This paper sets out:

- a reference architecture for an internal RAG assistant, component by component;
- the failure modes that most often degrade answer quality, and where each one originates;
- a practical evaluation checklist that separates retrieval quality from generation quality, so teams can find and fix problems rather than guess at them.

The central recommendation is to **measure retrieval and generation separately, from the first prototype onward**. Most answer-quality problems that look like model problems turn out to be retrieval problems, and they are far cheaper to fix once they can be seen.

---

## 1. Why Internal Knowledge Is a Hard Problem

Public question answering benefits from content that is written for strangers. Internal content is not. It is full of acronyms, project code names and assumptions that only make sense to the team that wrote it. It is also uneven: some documents are authoritative and current, others are drafts, and many are

quietly out of date.

Three properties of internal knowledge shape every design decision that follows:

1. **Permissions matter.** Not every employee may see every document. An assistant that surfaces a restricted HR file to the wrong person is a security incident, not a quality bug.
2. **Freshness matters.** Policies, prices and procedures change. An answer drawn from last year's version of a document can be worse than no answer.
3. **Provenance matters.** Employees need to know where an answer came from so they can check it and act on it with confidence.

A good RAG design treats all three as first-class requirements, not features to add later.

---

## 2. Reference Architecture

A production RAG system has two pipelines: an **offline pipeline** that prepares content, and an **online pipeline** that answers questions.

### 2.1 The offline pipeline: preparing the knowledge base

**Ingestion.** Connectors pull content from source systems along with its metadata: title, author, last-modified date, source location and access-control lists. Losing metadata at this stage is one of the most common and least visible mistakes, because later stages — permission filtering, freshness ranking, citations — all depend on it.

**Cleaning and parsing.** Raw content is converted to clean text. This is harder than it sounds. Tables, headers, footers, slide decks and scanned PDFs all need specific handling. Boilerplate such as navigation menus and legal footers should be stripped, or it will pollute retrieval.

**Chunking.** Documents are split into passages small enough to retrieve precisely but large enough to carry meaning on their own. Splitting along natural structure — sections, headings, paragraphs — generally works better than splitting at a fixed character count. Each chunk should keep a pointer to its parent document and, ideally, the heading path above it, so a passage about "eligibility" still knows it belongs to the travel policy rather than the benefits guide.

**Embedding and indexing.** Each chunk is converted into a vector by an embedding model and stored in an index alongside its text and metadata. Many teams also maintain a conventional keyword index, because exact terms such as product codes, error messages and names are often matched more reliably by keyword search than by semantic similarity.

### 2.2 The online pipeline: answering a question

**Query processing.** The user's question may be rewritten to resolve pronouns from earlier turns, expand acronyms or split a compound question into parts.

**Retrieval.** The system searches the index for candidate chunks. A hybrid approach — combining semantic vector search with keyword search — is a widely used default because each method catches what the other misses. **Permission filtering must happen here**, before any content reaches the model.

**Re-ranking.** A second, more precise model scores the candidates against the question and keeps only the best few. Re-ranking is often the single most effective upgrade to a basic pipeline.

**Prompt assembly.** The selected passages are inserted into a prompt with clear instructions: answer only from the provided sources, cite them, and say so plainly when the sources do not contain the answer.

**Generation and post-processing.** The model drafts the answer. Post-processing attaches citations as links to source documents and can apply checks — for example, verifying that each cited passage was actually among those retrieved.

---

### 3. Failure Modes

When a RAG assistant gives a bad answer, the symptom is the same — the user sees something wrong or unhelpful — but the cause can sit in any stage. The table below maps common failures to their origin and to the first fix worth trying.

| Failure mode                      | What the user sees                                                | Where it originates                                                     | First fix to try                                                             |
|-----------------------------------|-------------------------------------------------------------------|-------------------------------------------------------------------------|------------------------------------------------------------------------------|
| Missing content                   | "I couldn't find that," for a question the company has documented | Ingestion: source not connected, or parsing failed                      | Audit connector coverage; check parse output for key document types          |
| Retrieval miss                    | A vague or generic answer while the right document exists         | Retrieval: poor chunking, or the query and document use different terms | Hybrid search; structure-aware chunking; query rewriting                     |
| Wrong passage ranked first        | A confident answer about a related but different topic            | Ranking: similar-sounding passages outrank the correct one              | Add a re-ranker; include heading paths in chunks                             |
| Stale answer                      | Correct for an old version of a policy                            | Indexing: outdated documents not removed or down-weighted               | Incremental re-indexing; freshness signals in ranking                        |
| Unsupported claim (hallucination) | Plausible details that appear in no source                        | Generation: model fills gaps from its own training                      | Stricter grounding instructions; citation verification; allow "I don't know" |
| Conflicting sources               | An answer that blends two incompatible versions                   | Content: duplicates or contradictions in the knowledge base             | Mark authoritative sources; surface the conflict instead of merging it       |
| Permission leak                   | Content from a document the user should not see                   | Retrieval: access control applied too late or not at all                | Filter by access-control list at retrieval time, before generation           |
| Lost context                      | Good first answer, confused follow-ups                            | Query processing: conversation history not resolved                     | Rewrite follow-up questions into standalone queries                          |

Two lessons stand out from this list. First, most failures originate **before** the model generates a single word. Second, several failures — stale answers, conflicting sources — are really content-governance problems that RAG makes visible. An assistant cannot be more reliable than the documents behind it.

## 4. An Evaluation Checklist

Evaluating a RAG system with a single question — "Was the answer good?" — tells a team that something is wrong without telling them where. A better approach is to evaluate in layers.

## 4.1 Build a test set first

Assemble a set of realistic questions drawn from actual employee needs — help-desk tickets and search logs are good sources — and, for each, record the document or passage that contains the correct answer. Include questions that the knowledge base **cannot** answer; a system that never says "I don't know" will eventually invent something. Keep the set under version control and expand it whenever a real failure is reported.

## 4.2 Retrieval checks

- **Recall:** Is the correct passage among the retrieved candidates?
- **Ranking:** Is it near the top, within the set actually passed to the model?
- **Permissions:** Do test users with restricted access ever receive restricted passages? This check should be pass/fail with zero tolerance.
- **Freshness:** When a document has multiple versions, is the current one preferred?

## 4.3 Generation checks

- **Faithfulness:** Is every claim in the answer supported by the retrieved passages?
- **Citation accuracy:** Do the citations point to passages that actually support the statements they are attached to?
- **Answer relevance:** Does the answer address the question that was asked?
- **Abstention:** When the sources do not contain the answer, does the system say so?

## 4.4 Operational checks

- **Latency:** Is the end-to-end response time acceptable for the intended use?
- **Cost:** What does an average query cost across retrieval, re-ranking and generation?
- **Feedback loop:** Can users flag bad answers, and do those flags reach the team and the test set?

Automated scoring — including using a language model as a judge against the test set — makes these checks repeatable at scale, but it should be calibrated against human review of a sample. Automated judges have blind spots of their own.

---

# 5. Conclusion and Recommendations

Retrieval-augmented generation is the most practical way to put a conversational interface on top of an organization's private knowledge today. It keeps answers tied to sources the company controls, it respects the fact that knowledge changes, and it gives users a way to check what they are told. But its quality is set by its weakest stage, and most of those stages sit upstream of the model.

Teams building an internal RAG assistant should:

1. **Treat metadata as a core asset.** Preserve titles, dates, sources and access-control lists from ingestion onward; permissions, freshness and citations all depend on them.
2. **Enforce permissions at retrieval time.** Never rely on the model to withhold content it has already been given.
3. **Start with hybrid search and a re-ranker.** Together they address the most common retrieval failures at modest cost.
4. **Chunk along document structure** and carry heading context with every chunk.
5. **Build the test set before tuning.** Include unanswerable questions, and grow the set from every real failure.
6. **Measure retrieval and generation separately.** It is the fastest way to find out which stage to fix.
7. **Pair the system with content governance.** Assign owners to key documents, retire stale versions, and mark authoritative sources. The assistant will only ever be as trustworthy as the knowledge base behind it.

Organizations that follow these principles tend to move past the impressive demo to a tool employees actually rely on — which is the only measure of success that matters.

---

*Sample written for portfolio purposes; not client work. Original writing produced with an AI-assisted workflow and QA-checked.*