AI & LLMS

Retrieval-Augmented Generation in Production

Chunking, embeddings, vector search and reranking that actually

Houssam Kodad

PDF · DATAFORGE BOOKS

CONTENTS

Table of Contents

CHAPTER 01

Why RAG Breaks in Production

Most RAG systems work beautifully in the demo and disappoint in production, and the reason is almost always the same — and it is rarely the language model. It is retrieval. If the passage containing the answer never reaches the model, no amount of prompt engineering or model upgrading will produce a correct response. The model can only reason over what you give it, and giving it the right context is the hard, neglected part.

This chapter names that bottleneck plainly and lays out the full pipeline so you can see where quality leaks away. We walk through ingestion, chunking, embedding, retrieval, reranking and generation, and we catalogue the failure modes you will spend the rest of the book learning to diagnose and fix. The goal is to replace a vague sense that "the RAG isn't working" with a precise understanding of which stage is at fault.

Throughout, the emphasis is on rigorous evaluation of retrieval, not just of final answers. You cannot improve what you cannot measure, and most teams measure only the end-to-end output, which hides whether a bad answer came from bad retrieval or bad generation. We build the habit of measuring each stage, because that is what turns RAG from guesswork into engineering.

Why RAG Breaks in Production

When a RAG system gives a wrong or vague answer, the instinct is to blame the model or tweak the prompt. But in the large majority of cases the model never had a chance: the chunk containing the answer was not retrieved, or it was retrieved but buried beneath irrelevant context that drowned the signal. Garbage context in, garbage answer out — the oldest rule in computing applies with full force.

This reframes where effort should go. Retrieval quality — chunking, embeddings, search, reranking — is the lever that moves answer quality most, and it is also the part teams most often neglect in favour of prompt tinkering. The book is deliberately weighted toward retrieval for exactly this reason, because that is where the problems are and therefore where the improvements live.

The End-to-End Pipeline

A RAG pipeline has more moving parts than its simple description suggests, and a failure in any one of them quietly degrades the final answer. At ingestion, documents are chunked and embedded into a vector store. At query time, the question is embedded, candidates are retrieved — often by both keyword and vector search — reranked for precision, assembled into a context window, and finally

passed to the model with instructions to answer from that context.

Seeing the whole pipeline at once is what lets you trace a bad answer to its true cause instead of guessing. A failure at an early stage cannot be recovered at a later one: if retrieval misses the relevant chunk, no reranker or prompt can conjure it back. We keep this full pipeline in view throughout, because debugging RAG is fundamentally about isolating which link in the chain failed.

```
# The pipeline, and where quality silently leaks at each arrow:
# ingest:  docs -> chunk -> embed -> vector store
# query:   question -> embed -> retrieve (bm25 + vector)
#                   -> rerank -> assemble context -> LLM -> answer
#
# A miss at 'chunk' or 'retrieve' is invisible by the time you read the answer.
```

Chunking Without Losing Meaning

Chunking — splitting documents into retrievable pieces — looks trivial and is quietly decisive. Chunks that are too large dilute the relevant sentence among irrelevant ones and waste context budget; chunks that are too small lose the surrounding context needed to make sense of them. The right size and boundary depend on the content, and getting it wrong caps the quality of everything downstream.

Naive fixed-size chunking that splits mid-sentence or mid-table destroys meaning. Better strategies respect the document's structure — paragraphs, sections, headings — and sometimes overlap chunks so a fact near a boundary is not orphaned. We treat chunking as a first-class design decision rather than a default to accept, because it is one of the highest-leverage and most overlooked choices in the entire system.

Embeddings in Depth

Embeddings are how text becomes searchable by meaning rather than keywords: a model maps each chunk to a vector such that semantically similar text lands nearby in the space. The quality of this mapping sets a ceiling on retrieval — if the embedding model cannot tell that a question and its answer are related, no amount of clever indexing will retrieve the right chunk.

Choosing an embedding model is therefore a consequential decision, and the best general-purpose model is not always best for your domain. Specialised vocabulary — legal, medical, technical — often trips up generic embeddings, and domain adaptation or fine-tuning can dramatically improve retrieval. We cover how to evaluate embedding models on your own data rather than trusting a

leaderboard that was never measured on your problem.

Hybrid Retrieval and Reranking

Vector search captures meaning but can miss exact terms — a specific product code, an error number, a proper name — that keyword search nails. The robust answer is hybrid retrieval: run both keyword and vector search and combine their results, so you get semantic understanding and exact-match precision together. Each covers the other's blind spot, and the combination consistently beats either alone.

Reranking is the precision pass that follows. Retrieval is tuned for recall, casting a wide net to avoid missing the answer; a reranker then re-scores the candidates with a more expensive, more accurate model to push the truly relevant ones to the top. Because the model sees only the few passages that actually fit the context window, getting the most relevant ones to the very top is what makes or breaks the final answer.

Evaluating Retrieval and Answers

You cannot improve a RAG system without measuring it at two levels. Retrieval metrics ask whether the relevant chunks were found and ranked highly, independent of what the model then did with them. Answer metrics ask whether the final response is correct and faithful to the retrieved context. Measuring both is what lets you tell whether a bad answer came from missing context or from the model ignoring good context.

Faithfulness — whether the answer is actually supported by the retrieved passages, rather than invented — is the metric that matters most for trust, especially in domains where a confident fabrication is dangerous. We build evaluation sets and scoring methods for both retrieval and generation, because a RAG system without rigorous evaluation is a system you cannot safely change, and changing it is most of the work.

Keeping the Index Fresh

A RAG system is only as current as its index, and source data changes — documents are added, updated, and deleted. An index that drifts out of date answers confidently from stale information, which is often worse than admitting it does not know. Incremental indexing keeps the vector store synchronised with the source as content changes, without the cost of rebuilding everything constantly.

Deletion deserves special attention, both for correctness and increasingly for compliance: when a document must be removed, every chunk and embedding derived from it has to go too, or the system will keep surfacing content that should no longer exist. And when you change the embedding model, the entire index must be rebuilt to stay consistent. We treat index maintenance as the ongoing operational concern it is, because in production a RAG system is never finished — it is operated.

This is a free sample

You've reached the end of the sample chapter.

Get the complete book — every chapter, fully worked — at **dataforgebooks.com**.

FULL EDITION · 244 PAGES · PDF

Read the full title at **dataforgebooks.com**

Questions? support@dataforgebooks.com