title: Building Reliable Search & RAG Pipelines—Design, Tuning, and Failure Modes
theme: technology
subtopic: search-and-RAG
keywords: [ai, retrieval, embeddings, vector-stores, mmr, cosine]
approx_word_count: 980
suggested_sources:

* Wikipedia: Vector space model
* News/Report: Google Research Blog — “Retrieval-Augmented Generation for Knowledge-Intensive NLP”

Building Reliable Search & RAG Pipelines—Design, Tuning, and Failure Modes

Overview

Retrieval-augmented generation (RAG) couples search with language generation so that answers cite evidence. When engineered carefully, RAG improves factuality and keeps proprietary knowledge in-house. This briefing lays out end-to-end pipeline patterns, explains **MMR vs. pure similarity** retrieval, and catalogs common breakpoints with concrete diagnostics.

The Pipeline at a Glance

1) Load

Ingest content from the web, internal wikis, PDFs, tickets, and databases. Normalize to a common schema: `id`, `source`, `timestamp`, and raw text/HTML. Extract tables and images with OCR if needed.

2) Split

Chunk text to keep context coherent and token budgets under control. Practical defaults:

* **Size**: 300–800 tokens for LLMs; shorter for terse FAQs.
* **Overlap**: 10–20% to protect sentence boundaries.
* **Structure-aware**: split on headings and bullet boundaries; keep tables intact.
* **Metadata carryover**: preserve source URL, section, and timestamps for later

citation and recency filters.

3) Embed

Choose an embedding model with strong multilingual and domain performance. Store vectors (e.g., 384–1024 dims). Track model version to allow **re-indexing** when models improve.

4) Store

Index vectors in a **vector store** (HNSW, IVF-PQ, or ScaNN-based) with metadata filters. Also keep a **lexical index** (BM25) for keywords, numeric strings, and exact code tokens.

5) Retrieve

At query time:

* **Lexical pass**: Top-k via BM25 for exact match.
* **Vector pass**: ANN search by **cosine** (or dot-product) similarity.
* **Hybrid fusion**: Reciprocal rank fusion (RRF) or score normalization to merge lists.
* **MMR re-ranking**: Enforce diversity by penalizing near duplicates among the chosen chunks.

6) Augment & Generate

Compose a prompt with the query, the top-N evidence chunks, and explicit instructions: “Answer **only** from the provided context; if missing, say you don’t know. Cite sources.” Pass to a response model sized for latency and cost.

7) Grounding & Post-Processing

* **Attribution**: Inline citations with titles and anchors.
* **Extraction**: For forms or tables, parse with regex/grammar to avoid free-form errors.
* **Safety & Policy**: Validate outputs against allowlists (e.g., only certain commands/actions).

MMR vs. Similarity—When Diversity Beats “Closest”

Pure similarity returns the **closest** chunks to the query vector. That can be suboptimal when those chunks say the **same thing**. **MMR (Maximal Marginal Relevance)** selects items that are relevant **and** dissimilar to already selected items. Benefits:

* **Coverage**: multiple viewpoints or sections get represented.
* **Reduced redundancy**: fewer near-duplicate chunks that waste token budget.
* **Fewer blind spots**: especially for multi-facet queries (“pricing AND SLA”).

A practical recipe: take top-50 by cosine, then greedily build a top-8 using MMR with $\lambda \approx 0.7$ (tune on validation). In many corpora, this boosts retrieval **recall@8 by ~5–10%** with the same context window.

Knobs That Matter

* **Chunk length**: Long chunks improve recall but waste tokens; short chunks are precise but brittle. Try 400–600 tokens with 50–80 token overlap as a baseline.
* **k values**: Retrieval depth of 40–100 before re-ranking is a good starting region.
* **Hybrid strength**: For code or numeric queries, up-weight BM25; for semantic questions, lean on embeddings.
* **Freshness**: Filter by timestamp (e.g., `>= last_90_days`) or apply decay in scoring.
* **Query rewriting**: Expand acronyms, correct spelling, add synonyms (“SLA”→“service level agreement”).

Failure Modes and How to Detect Them

1) Hallucination Despite Good Retrieval

Symptom: The model cites sources but invents details.
Causes: Prompt too open-ended; the model is asked to synthesize beyond evidence; numeric reasoning errors.
Fixes: Tighten instructions, use **extractive QA** for numbers, add **tool-checks** (e.g., calculator), and shorten the number of evidence chunks if the model is getting distracted.

2) Retriever Miss—Answer Exists but Wasn’t Fetched

Symptom: Ground truth document exists, not present in top-k.
Causes: Poor chunking, stale embeddings, query mismatch, or missing synonyms.
Fixes: Re-index with a newer embedding model; add **query expansion**; apply **MMR**; add a lexical union; increase initial k.

Diagnostic: Compute **oracle recall** (does the answer appear in top-100 by any method?) and **recall@k** on a labeled set.

3) Over-Redundant Context

Symptom: Many near-duplicate chunks; model repeats the same lines.
Causes: Highly templated docs; lack of diversity controls.
Fixes: Apply **MMR**; de-duplicate by content hash; reduce per-source quotas.

4) Poisoned or Low-Quality Chunks

Symptom: Model parrots outdated or adversarial text copied into wikis.
Fixes: Trust scores per source, automated quality filters (language detection, toxicity), and **blocklists**. Run **age-based decay** to down-weight old content.

5) Tool/Index Drift

Symptom: Sudden recall drop after deployment.
Causes: Embedding model version change; ANN parameters tweaked; schema mismatch.
Fixes: Version every component. Add **canary queries** and **dashboards** for recall@k, hit rate, and MMR coverage.

Evaluating RAG End-to-End

* **Retrieval**: recall@k, precision@k, MRR.
* **Groundedness**: percent of generated claims supported by cited text (LLM or rules-based grader).
* **Answer Quality**: task-specific metrics (exact match, ROUGE-L), plus human side-by-side.
* **Latency/Cost**: p95 latency, tokens per answer, cache hit rates.

* **Safety**: refusal on out-of-scope questions; leakage tests (no internal secrets in outputs).

A disciplined approach uses **frozen benchmarks** (e.g., 300 labeled questions) and a weekly regression suite. Track not only averages but **tails**—worst-10% groundedness is often where risk lives.

Operating the System

* **Caching**: memoize embeddings and retrieval results for popular queries.
* **Sharding**: split vector indexes by tenant or topic; route queries by metadata.
* **Observability**: log which chunks were retrieved, the scores, and the final prompt.
* **Privacy**: keep proprietary corpora in isolated indexes; redact secrets before storage.

Beyond the Basics

* **Rerankers**: cross-encoders that score query–document pairs can improve precision@10 by **5–15%** with a modest latency trade-off.
* **Agents with Tools**: let the system issue follow-up retrievals (“drill down on pricing terms”).
* **Multimodal RAG**: images and tables embedded alongside text; handy for handbooks and diagrams.

Key Takeaways

* Treat RAG as an engineering system: load → split → embed → store → retrieve → generate → verify.
* Pure similarity is a baseline; **MMR** usually improves coverage and reduces redundancy.
* Most failures trace to chunking, embeddings, or hybrid retrieval weights—not the LLM.
* Evaluate retrieval, groundedness, and answer quality separately to avoid chasing noise.
* Version everything and watch recall@k like a service-level objective.