
🔍 RAG Retrieval: Full Breakdown

🚨 What Is Retrieval?

In RAG, **retrieval** means fetching relevant chunks from your vector database (like FAISS, Chroma, Pinecone, etc.) using a user's query — so your LLM has real, grounded info to work with.

📦 What's Typically Involved in Retrieval?

1. **Vector store selection & loading**
2. **Embedding search (similarity search / MMR)**
3. **Custom retriever logic (filters, rerankers, etc.)**
4. **Query augmentation or refinement (optional)**
5. **Return top-k relevant documents**
6. **Pass results into prompt for generation**

🧠 First: Setup Recap

- Converted docs → chunks
- Generated embeddings
- Indexed them in something like FAISS or Chroma

I'll continue using **FAISS + HuggingFace + LangChain** for this walkthrough (can swap out for Chroma or Pinecone if needed).

🔍 EXPLAINING EVERY RETRIEVAL METHOD

1️⃣ Basic Retrieval (basicRetrieval)

What:
Classic vector search using **cosine similarity** under the hood. Returns top k most similar document chunks.

When to use:

- Your data is well-embedded.
- You want speed and simplicity.
- You're okay with a few irrelevant chunks sometimes.

How it works:

1. Query → embedding
2. Search vector store using cosine similarity
3. Return top k matches

⚠️ Downside: may return similar-sounding junk without meaning diversity.

2️⃣ MMR Retrieval (mmrRetrieval)

What:
Max Marginal Relevance: balances similarity with **diversity** in results.

When to use:

- You're getting redundant/too-similar answers.
- You want coverage of different angles (e.g. definitions + examples).

How it works:

- Starts with the top most similar doc
- Adds the next one that's both relevant **and** not redundant
- Repeats until k is reached

🤝 Great when user asks: "Tell me about pros & cons of X."

3️⃣ LangChain Retriever (getRetriever)

What:
Turns a vector store into a **LangChain-compatible retriever**. It's just a clean wrapper.

When to use:

- When you want to plug the retriever into LangChain chains (like RetrievalQA)
- You want to keep config (like k) clean and modular

How it works:

- Uses .as_retriever() to abstract away vector DB logic
- Lets you call .get_relevant_documents(query)

🧱 It's just infra setup — no change in logic.

4️⃣ Retrieval QA Chain (runRetrievalQA)

What:
Combines **retriever + LLM** into a question-answering pipeline.

When to use:

- You want your RAG system to give direct natural language answers.
- You want to return **answers**, not just docs.

How it works:

1. Query → retriever → top docs
2. Those docs are passed into a prompt
3. LLM generates an answer from them

🔥 Core RAG use case.

5️⃣ Query Expansion (expandQuery)

What:
Uses LLM to rewrite vague queries into more specific, search-friendly versions.

When to use:

- Users type low-effort stuff like "Tell me about lungs"

- You want to **increase retrieval accuracy**

How it works:

- LLM prompt reformulates the input
- New query is then used for retrieval

🧠 Think of it like a brainy autocomplete before the search.

6️⃣ Retrieval With Score Threshold (retrievalWithScoreThreshold)

What:
Filters out results that don't meet a **minimum similarity score**.

When to use:

- You want to **avoid hallucinations**
- You only want "high-confidence" chunks used in generation

How it works:

- After retrieval, compare each chunk's similarity score
- Only return those with score > threshold (e.g. 0.8)

🧽 Helps clean the garbage from your vector DB when recall is too wide.

7️⃣ Custom Similarity Search (customSimilaritySearch)

What:
Manually computes cosine similarity instead of letting vector DB handle it — lets you customize scoring logic.

When to use:

- You want to experiment with new scoring methods
- You don't trust the default ranking logic

How it works:

- Get embedding for query + chunks
- Use sklearn to compute cosine similarity manually

- Sort and return top-scoring chunks

🛠️ Great for research-level tweaking or diagnostics.

8️⃣ Hybrid Retrieval (hybridRetrieval)

What:
Combines **vector search** with **BM25 (keyword-based)** search. Ensemble of both.

When to use:

- Your documents contain formulas, code, names (bad for embedding-only)
- You want the best of both worlds (semantic + literal)

How it works:

- Vector retriever ranks by similarity
- BM25 retriever ranks by keyword match
- Combines scores using weights (e.g., 0.7 vector + 0.3 BM25)

🧬 Very common in production RAG.

9️⃣ Cohere Re-Ranking (rerankWithCohere)

What:
After top k results are retrieved, uses Cohere's model to **re-rank** them by true relevance.

When to use:

- Your top-5 chunks are often just "kinda" related
- You want stronger semantic understanding at ranking time

How it works:

- For each doc, sends query + doc to Cohere reranker
- Gets back a new score
- Resorts the list by score

🧠 Use this when precision matters more than speed.

🧠 Mnemonic to Remember When to Use What

"BMM-Q-HCRC"
(**B**asic, **M**MR, **M**etadata, **Q**ueryExpand, **H**ybrid, **C**ustom, **R**erank, **C**ohere)

- 🐢 Basic → just get started
- 🌀 MMR → remove redundancy
- 🎯 Threshold → reduce junk
- 🧠 QueryExpand → fix vague queries
- 🧬 Hybrid → mix meaning + keywords
- 🧪 Custom → test your own logic
- 🧱 Rerank → boost precision
- 🔥 Cohere → elite reranking
