

OFFICIAL COURSE STUDY GUIDE

RAG From Scratch

Master Retrieval-Augmented Generation from the team behind LangChain.

AI & AGENTIC ENGINEERING · INTERMEDIATE · 4 MODULES · 1 HR 34 MIN

Course by LangChain · Curated and published by MegaCourse

MEGACOURSE.APP

RAG From Scratch: The Complete Study Guide

2026 Edition

Published by MegaCourse · megacourse.app

Learn Anything. Achieve Everything.

MegaCourse curates the world's best free courses into structured learning paths with study guides, practice material, and certificates. This guide accompanies the MegaCourse course "RAG From Scratch", taught by LangChain. It is designed to be read alongside the video lessons and to serve as a standalone reference for review.

Study guide text copyright © 2026 MegaCourse. Course video content remains the property of its creators. You may print this guide for personal study. Redistribution or resale is not permitted.

Enroll, track progress, and earn your certificate at
megacourse.app/courses/rag-from-scratch

Contents

<i>About This Course</i>	4
CHAPTER 1 Understanding Retrieval-Augmented Generation	6
CHAPTER 2 Advanced Retrieval Techniques	12
CHAPTER 3 RAG Reference and Pitfalls	17
CHAPTER 4 Final Assessment: RAG	21
<i>Glossary</i>	23
<i>Answer Key</i>	25
<i>Where to Go Next</i>	29

INTRODUCTION

About This Course

Welcome to the study guide for "RAG From Scratch", a comprehensive course designed to help you master Retrieval-Augmented Generation. This crucial technique improves the reliability and accuracy of Large Language Model applications by grounding their responses in external knowledge.

The course is structured into 4 modules and 17 lessons, providing about 1 hour and 34 minutes of direct instruction. It is taught by the team behind LangChain, ensuring you learn from leading experts in the field.

MegaCourse has enhanced the original material by adding curated readings, a final knowledge-check quiz, and a clear learning path. This guide complements the video lessons, offering a valuable resource for review and deeper understanding.

Course Facts

FORMAT	4 modules · 17 lessons
VIDEO INSTRUCTION	About 1 hr 34 min
LEVEL	Intermediate
CATEGORY	AI & Agentic Engineering
INSTRUCTOR	LangChain
CERTIFICATE	Included with course completion on MegaCourse
COURSE HOME	megacourse.app/courses/rag-from-scratch

What You Will Learn

- Understand the full RAG pipeline from end to end.
- Chunk, embed, and index documents effectively.
- Build functional retrieval and re-ranking systems.
- Reduce hallucinations in LLM applications.

Who This Course Is For

This course is ideal for developers and AI engineers who want to build more reliable and accurate applications using Large Language Models. If you aim to reduce hallucinations and provide LLMs with specific, up-to-date information, this course will equip you with essential skills.

Before you begin, you should be proficient in Python and have a basic familiarity with how Large Language Models operate.

Before You Start

- Proficiency in Python.
- Basic familiarity with Large Language Models.

How to Use This Guide

This study guide is designed to work alongside the video lessons. We recommend watching each video lesson first to grasp the core concepts. Then, use this guide to review key ideas, work through examples, and reinforce your understanding.

The course is organized into 4 modules and 17 lessons. Pace yourself by completing one module at a time, ensuring you understand the material before moving on. The guide includes summaries and additional explanations to help clarify complex topics.

Use the included knowledge checks and review questions to test your comprehension. A glossary of terms is also provided to help you navigate new terminology.

CHAPTER 1

Understanding Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is a powerful technique for improving the factual accuracy and relevance of large language model (LLM) outputs. LLMs are trained on vast datasets, but their knowledge is static and can become outdated. They also sometimes "hallucinate", generating plausible but incorrect information. RAG addresses these limitations by connecting LLMs to external, up-to-date knowledge bases.

This chapter introduces the fundamental components of the RAG pipeline. You will learn how to prepare your data for retrieval, execute effective searches, and combine retrieved information with an LLM's generative capabilities. We then explore advanced strategies for query translation, which enhance retrieval effectiveness by rephrasing or expanding user questions. Mastering these concepts is essential for building robust and reliable LLM applications.

The Core RAG Pipeline: Indexing, Retrieval, and Generation

Covers: RAG From Scratch: Part 1 (Overview) · RAG From Scratch: Part 2 (Indexing) · RAG From Scratch: Part 3 (Retrieval) · RAG From Scratch: Part 4 (Generation)

Retrieval-Augmented Generation (RAG) integrates a retrieval system with a large language model (LLM) to enhance the LLM's ability to generate accurate and contextually relevant responses. The core RAG pipeline consists of three main stages: indexing, retrieval, and generation. This structured approach helps LLMs access specific, up-to-date information that was not part of their original training data, thereby reducing hallucinations and improving factual grounding.

Indexing: Preparing Your Knowledge Base

The first stage, indexing, prepares your external data source for efficient searching. This process involves converting raw documents into a format that a computer can quickly understand and compare.

1. **Document Loading:** Begin by loading your raw data. This could be text files, PDFs, web pages, or database records. Each document is typically treated as a unit of information.
2. **Chunking:** Large documents are often too long to fit into an LLM's context window or to be effectively searched. Chunking breaks these documents into smaller, manageable segments. A common strategy is to split text by paragraphs or sentences, or to use fixed-size chunks with some overlap. Overlap helps maintain context across chunk boundaries. For example, a document might be split into chunks of 500 characters with 50 characters of overlap.
3. **Embedding:** Each text chunk is then converted into a numerical representation called a vector embedding. An embedding model processes the text and outputs a dense vector, a list of numbers, that captures the semantic meaning of the chunk. Chunks with similar meanings will have embeddings that are numerically close to each other in a high-dimensional space.

4. **Vector Store:** Finally, these vector embeddings are stored in a specialized database known as a vector store. The vector store allows for rapid similarity searches. Along with the vector, the original text chunk and any associated metadata (like document title or source URL) are also stored.

Consider a practical example. You have a company policy document.

- You load the PDF.
- You chunk it into 200-word segments, ensuring a 20-word overlap.
- Each segment is then passed to an embedding model, such as `text-embedding-ada-002`, to generate a vector.
- These vectors, along with their corresponding text segments, are stored in a vector database like Chroma or Pinecone.

Retrieval: Finding Relevant Information

When a user asks a question, the retrieval stage finds the most relevant chunks from your indexed knowledge base.

1. **Query Embedding:** The user's query is first converted into a vector embedding using the *same* embedding model used during indexing. This ensures that the query and document chunks are represented in the same semantic space.
2. **Similarity Search:** The query's embedding is then compared against all the document chunk embeddings in the vector store. A similarity metric, such as cosine similarity, is used to calculate how "close" each chunk's embedding is to the query's embedding.
3. **Top-K Selection:** The retrieval system returns the `k` most similar document chunks. `k` is a configurable parameter, typically between 3 and 10. These `k` chunks are considered the most relevant context for answering the user's question.

If a user asks, "What is the policy on remote work?", the system embeds this question. It then searches the vector store for chunks whose embeddings are most similar to the query embedding. It might retrieve policy sections on "flexible work arrangements", "home office guidelines", and "remote employee benefits".

Generation: Crafting the Answer

The final stage combines the retrieved context with the LLM to generate a coherent and accurate answer.

1. **Prompt Construction:** The retrieved `k` chunks of text are inserted into a carefully designed prompt. This prompt typically instructs the LLM to answer the user's question *using only the provided context*. This instruction is critical for preventing the LLM from relying on its internal, potentially outdated, knowledge.

```
"You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know.
Question: {question}"
```

```
Context: {context}
Answer:"
```

Here, `{question}` is the user's original query, and `{context}` is the concatenation of the retrieved text chunks. **2. LLM Inference:** The constructed prompt is sent to the LLM. The LLM processes the prompt and generates a response based on the provided context.

By following this pipeline, RAG ensures that the LLM's output is grounded in verifiable external information, significantly improving its reliability for factual queries. Common mistakes include using an embedding model for queries different from the one used for indexing, or not explicitly instructing the LLM to stick to the provided context. Always verify that your chunking strategy preserves meaning and that your `k` value retrieves sufficient, but not overwhelming, context.

Advanced Retrieval: Query Translation Techniques

Covers: RAG from scratch: Part 5 (Query Translation -- Multi Query) · RAG from scratch: Part 6 (Query Translation -- RAG Fusion) · RAG from scratch: Part 7 (Query Translation -- Decomposition) · RAG from scratch: Part 8 (Query Translation -- Step Back)

While the basic RAG pipeline is effective, user queries can sometimes be ambiguous, complex, or phrased in a way that makes direct similarity search less effective. Query translation techniques address these challenges by transforming the original user query into one or more improved queries before retrieval. This preprocessing step significantly enhances the quality of the retrieved context.

Multi-Query

The Multi-Query approach generates several distinct rephrased versions of the original user query. This strategy acknowledges that a single query might not perfectly capture all relevant aspects of a user's intent. By creating multiple perspectives, the system increases the chances of finding relevant documents, even if they use different phrasing.

1. **Generate Variations:** An LLM is prompted to generate 3 to 5 alternative questions that are semantically similar to the original query. *Original Query:* "What are the benefits of a plant-based diet?" *LLM Generated Queries:*

- "How does a vegan diet impact health?"
- "What are the advantages of eating only plants?"
- "Describe the health outcomes associated with vegetarianism."

2. **Execute Multiple Retrievals:** Each generated query is then used to perform a separate retrieval operation against the vector store. This results in multiple sets of retrieved documents.

3. **Combine and Deduplicate:** All retrieved documents from these separate searches are combined. Duplicate documents are removed, and the remaining unique documents form a richer, more comprehensive context for the generation phase.

This method is particularly useful for broad or slightly ambiguous questions, as it casts a wider net during retrieval.

RAG Fusion

RAG Fusion combines the Multi-Query approach with a re-ranking mechanism to prioritize the most relevant documents. Instead of simply concatenating all retrieved documents, RAG Fusion uses Reciprocal Rank Fusion (RRF) to score and order the results.

1. **Multi-Query Generation and Retrieval:** Similar to Multi-Query, several queries are generated, and separate retrievals are performed.
2. **Reciprocal Rank Fusion (RRF):** For each retrieved document, its rank across all the individual query results is considered. RRF assigns a score to each document based on its position in the ranked lists. Documents that appear high in multiple lists receive a higher RRF score. *Formula:* $RRF_Score = \sum (1 / (rank + k))$ for each query a document appears in, where k is a constant (often 60) to prevent division by zero for rank 0 and smooth scores.
3. **Top Document Selection:** The documents are then re-ranked according to their RRF scores, and the top k documents are selected for generation.

RAG Fusion is effective because it uses the diversity of multiple queries while using a robust statistical method to identify truly salient documents. It provides a more refined selection than simple concatenation.

Decomposition

Complex user queries often contain multiple sub-questions or require information from different parts of the knowledge base. Decomposition breaks down a single, complex query into several simpler, atomic sub-queries.

1. **Decompose Query:** An LLM analyzes the original query and identifies its constituent parts, generating a list of simpler questions. *Original Query:* "What are the capital cities of France and Germany, and what are their primary languages?" *LLM Decomposed Queries:*
 - "What is the capital city of France?"
 - "What is the primary language of France?"
 - "What is the capital city of Germany?"
 - "What is the primary language of Germany?"
2. **Retrieve for Each Sub-query:** Each sub-query is then used to perform an independent retrieval.
3. **Synthesize Answer:** The retrieved contexts for all sub-queries are combined, and the LLM uses this comprehensive context to answer the original complex question.

Decomposition ensures that all aspects of a complex query are addressed, preventing the retrieval system from focusing on only one part of the question.

Step-Back Prompting

Step-Back Prompting encourages the LLM to identify the high-level, conceptual question underlying a specific user query. Instead of directly answering the specific question, the LLM first "steps back" to a

more general principle or concept, retrieves information related to that general concept, and then uses that broader context to answer the original specific question.

1. **Identify Step-Back Question:** An LLM is prompted to extract a more abstract, general question from the user's specific query. *Original Query:* "What is the optimal temperature for storing fresh produce to maximize shelf life?" *LLM Step-Back Question:* "What factors affect the shelf life of fresh produce?"
2. **Retrieve General Context:** The step-back question is used for retrieval, yielding documents related to the broader concept.
3. **Answer Specific Question:** The LLM then uses this general context, along with the original specific query, to formulate a precise answer.

This technique is useful when the specific answer might be implicitly contained within broader knowledge, or when the user's specific question requires understanding a general principle first. For example, knowing the general factors affecting shelf life (humidity, light, ethylene) helps determine optimal temperature.

Query translation methods significantly improve RAG system performance by making retrieval more robust and adaptable to various query types. Experiment with these techniques to find which best suits your specific application and data.

KEY TAKEAWAYS

- Retrieval-Augmented Generation (RAG) enhances LLM accuracy by grounding responses in external, up-to-date knowledge.
- The core RAG pipeline involves indexing documents, retrieving relevant chunks, and generating an answer using an LLM.
- Indexing converts raw data into searchable vector embeddings stored in a vector database.
- Retrieval matches user query embeddings to document chunk embeddings to find the most similar context.
- Query translation techniques like Multi-Query, RAG Fusion, Decomposition, and Step-Back Prompting improve retrieval effectiveness for complex or ambiguous questions.
- Multi-Query generates multiple query variations, RAG Fusion re-ranks results, Decomposition breaks down complex questions, and Step-Back identifies underlying concepts.

CHECK YOUR UNDERSTANDING

Answer these before moving on. Model answers are in the Answer Key at the back of this guide.

1. Describe the three main stages of the core RAG pipeline and their purpose.
2. Explain the role of chunking and embedding in the indexing stage. Why are both necessary?
3. How does Multi-Query differ from a single-query retrieval, and what problem does it aim to solve?

4. What is Reciprocal Rank Fusion (RRF) and how is it used in RAG Fusion?
5. When would you use Decomposition as a query translation technique? Provide an example.

CHAPTER 2

Advanced Retrieval Techniques

This chapter explores advanced strategies for improving the retrieval component of Retrieval-Augmented Generation (RAG) systems. Effective retrieval is crucial for reducing hallucinations and providing accurate, contextually relevant information to a Large Language Model (LLM). We will cover methods for understanding and transforming user queries, as well as sophisticated techniques for indexing and retrieving documents.

You will learn how to go beyond simple keyword matching and basic vector search. The techniques presented here enable RAG systems to handle complex queries, diverse data sources, and large document collections with greater precision and recall. Mastering these methods will significantly enhance the performance and reliability of your LLM applications.

Enhancing Query Understanding

Covers: RAG from scratch: Part 9 (Query Translation -- HyDE) · RAG from scratch: Part 10 (Routing) · RAG from scratch: Part 11 (Query Structuring)

The initial step in any RAG pipeline is to understand the user's query. A well-understood query leads to more relevant document retrieval. This section introduces three powerful techniques to enhance query understanding: Hypothetical Document Embedding (HyDE), Query Routing, and Query Structuring. Each method transforms or interprets the original query to improve the subsequent retrieval process.

Hypothetical Document Embedding (HyDE)

HyDE addresses the challenge where a user's query might not be semantically similar to the relevant documents, even if they discuss the same topic. Traditional embedding models might struggle to find a match. HyDE overcomes this by first asking an LLM to generate a *hypothetical answer* or document based on the user's query. This generated document is then embedded, and its embedding is used to search the vector database. The rationale is that a hypothetical answer, even if imperfect, is often semantically closer to actual relevant documents than the original short query.

For example, if a user asks, "What are the key features of the new smartphone model X?", an LLM might generate a paragraph describing typical smartphone features, including processor, camera specifications, and battery life. This longer, more descriptive text will likely have a higher semantic similarity to actual product reviews or specification sheets in the index than the short query itself.

To implement HyDE, you typically perform these steps:

1. Receive the user's query.
2. Pass the query to an LLM, prompting it to generate a plausible, hypothetical answer or document.
3. Embed this hypothetical document using your chosen embedding model.
4. Use this embedding to perform a vector search against your document index.

5. Retrieve the top-k documents and pass them, along with the original query, to the final LLM for generation.

A common mistake is to over-rely on the LLM's hypothetical answer for factual accuracy. Remember, the generated document is a *tool for retrieval*, not the final answer. Its purpose is to guide the search to relevant real documents.

Query Routing

Query routing involves directing a user's query to the most appropriate data source or tool. This is particularly useful in systems with multiple specialized knowledge bases, APIs, or tools. Instead of searching everything, the system first determines the *intent* of the query.

Consider a RAG system that has access to both a product catalog database and a customer support FAQ document store. A query like "How do I reset my password?" should be routed to the FAQ, while "What is the price of product Y?" should go to the product catalog. A router component, often another LLM or a classification model, analyzes the query and decides the destination.

The router can be implemented using:

- **LLM-based routing:** An LLM is prompted to output a specific tool name or data source identifier based on the query.
- **Semantic routing:** Embeddings of the query are compared to embeddings of descriptions of available tools or data sources.
- **Keyword-based routing:** Simple keyword matching can direct queries, though this is less robust.

Effective routing reduces the search space, improves retrieval speed, and prevents irrelevant information from reaching the LLM, thus reducing potential hallucinations.

Query Structuring

Query structuring transforms a natural language query into a structured format, such as a SQL query, a Cypher query for graph databases, or an API call. This allows the RAG system to interact with structured data sources directly, retrieving precise answers rather than relying solely on semantic search over unstructured text.

For example, a user query "Show me all customers who purchased product A in the last month" could be transformed into a SQL query:

```
SELECT * FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.product_id = 'product_A'
AND Orders.order_date >= DATE('now', '-1 month');
```

This structured query can then be executed against a database. The results are retrieved and can be used as context for the LLM.

Steps for query structuring often involve:

1. Identifying the intent and entities in the natural language query.

2. Using an LLM, fine-tuned model, or rule-based system to generate the structured query.
3. Executing the structured query against the relevant data source.
4. Formatting the results for the LLM.

Query structuring is powerful for factual questions where data resides in structured formats. It ensures accuracy and avoids the ambiguity inherent in purely semantic retrieval.

Advanced Indexing and Retrieval Strategies

Covers: RAG From Scratch: Part 12 (Multi-Representation Indexing) · RAG From Scratch: Part 13 (RAPTOR) · RAG From Scratch: Part 14 (ColBERT)

Beyond understanding the query, how documents are indexed and retrieved plays a critical role in RAG performance. This section introduces advanced strategies for indexing and retrieval that go beyond simple chunking and single-vector embeddings. These methods aim to capture more nuanced information, handle complex document structures, and improve the efficiency and accuracy of the search process.

Multi-Representation Indexing

Standard RAG pipelines often index document chunks as single vectors. Multi-representation indexing addresses the limitation that a single vector might not fully capture all relevant aspects of a document or its various uses. This technique involves creating multiple representations for each document or chunk and indexing them.

Common approaches include:

- **Original Chunk + Summary:** Indexing the original document chunk alongside a concise summary of that chunk. The summary might be better for high-level semantic matching, while the original chunk provides detailed context.
- **Original Chunk + Question-Answer Pairs:** Generating several question-answer pairs from a chunk and indexing these pairs. A user's question might match one of these generated questions more directly.
- **Different Embedding Models:** Indexing the same chunk with embeddings from multiple different models. This provides robustness, as different models might excel at different types of queries.
- **Hierarchical Indexing:** Indexing documents at various granularities (e.g., entire document, sections, paragraphs, sentences). This allows retrieval at the most appropriate level of detail.

When a query comes in, the system can choose which representation to search against, or search multiple representations and aggregate results. For example, a query could first search summaries for broad topics, then use the retrieved summaries to identify specific original chunks for detailed information. This approach improves recall and the relevance of retrieved context.

RAPTOR (Recursive Abstractive Processing for Tree-Organized Retrieval)

RAPTOR is an advanced indexing strategy designed to handle very long documents or collections by creating a hierarchical, tree-like structure of document representations. It addresses the problem of losing context when documents are simply split into small, independent chunks.

The core idea of RAPTOR is:

1. **Initial Chunking:** Break documents into small, manageable chunks.
2. **Hierarchical Clustering:** Cluster these chunks based on semantic similarity.
3. **Abstractive Summarization:** For each cluster, generate a concise summary that captures the main ideas of its constituent chunks.
4. **Recursive Processing:** Treat these summaries as new "documents" and repeat steps 2 and 3, creating higher-level summaries. This process continues until a root summary or a desired hierarchy depth is reached.

The result is a tree where leaf nodes are original chunks, and internal nodes are summaries of their children. When a query comes in, the system can traverse this tree, starting from higher-level summaries, to quickly identify relevant sections before drilling down to the specific original chunks. This allows for both broad topic matching and precise detail retrieval.

RAPTOR is particularly effective for large, complex knowledge bases where a user's query might relate to a high-level concept that spans multiple low-level chunks. It provides a more coherent and context-rich retrieval experience.

ColBERT (Contextualized Late Interaction over BERT)

ColBERT is a powerful retrieval model that combines the strengths of dense retrieval (semantic understanding) with the precision of lexical matching. Unlike traditional dense retrievers that produce a single vector for a query and a single vector for a document, ColBERT generates multiple contextualized embeddings for each query token and each document token.

The "late interaction" part is key. Instead of calculating a single dot product between a query vector and a document vector, ColBERT performs *token-level similarity comparisons* between query token embeddings and document token embeddings. For each query token, it finds the most similar document token embedding. These maximum similarities are then summed up to get a final relevance score for the document.

This approach offers several benefits:

- **Granular Matching:** It allows for fine-grained matching between specific parts of the query and specific parts of the document.
- **Contextual Understanding:** The embeddings are contextualized by BERT, capturing semantic meaning.
- **Efficiency:** While it involves more comparisons than single-vector methods, it is designed to be efficient for retrieval, often using specialized indexing structures (like FAISS) that can handle the multiple embeddings.

ColBERT provides a strong balance between semantic relevance and keyword precision, often outperforming simpler dense retrieval models, especially for complex or multi-faceted queries. It ensures that not only the overall topic but also specific terms and phrases are considered during retrieval.

KEY TAKEAWAYS

- HyDE improves retrieval by generating hypothetical documents whose embeddings are more semantically aligned with relevant content than the original query.
- Query routing directs queries to specific data sources or tools based on intent, optimizing search and reducing irrelevant context.
- Query structuring transforms natural language into executable formats like SQL, enabling precise interaction with structured databases.
- Multi-representation indexing stores documents in various forms, such as summaries or Q&A pairs, to enhance retrieval flexibility and accuracy.
- RAPTOR builds a hierarchical index of document summaries, allowing for multi-granular retrieval from broad topics to specific details.
- ColBERT uses late interaction to perform token-level similarity comparisons, balancing semantic understanding with lexical precision for improved retrieval.

CHECK YOUR UNDERSTANDING

Answer these before moving on. Model answers are in the Answer Key at the back of this guide.

1. Explain the core idea behind Hypothetical Document Embedding (HyDE) and why it can improve retrieval performance.
2. Describe a scenario where Query Routing would be beneficial in a RAG system.
3. How does Query Structuring differ from traditional semantic search in a RAG pipeline?
4. What is the main advantage of Multi-Representation Indexing, and provide two examples of different representations.
5. Briefly explain how RAPTOR helps in handling long documents more effectively than simple chunking.
6. What is "late interaction" in the context of ColBERT, and why is it important for retrieval?

CHAPTER 3

RAG Reference and Pitfalls

This chapter provides a comprehensive reference for Retrieval-Augmented Generation (RAG) systems. You will learn the complete RAG pipeline, step by step, and understand the five critical levers that determine its quality. Mastering these elements is essential for building effective RAG applications.

Beyond the fundamentals, this chapter also focuses on identifying and resolving common failure points in RAG. We will explore why RAG systems often fail to deliver accurate answers and equip you with practical strategies and tools to diagnose and fix these issues, ensuring your LLM applications provide reliable information.

The RAG Pipeline, End to End

Covers: The RAG Pipeline, End to End

Retrieval-Augmented Generation (RAG) is a powerful technique that grounds a large language model (LLM) in your specific data. This process ensures the LLM answers questions using verifiable facts from your documents, rather than relying solely on its pre-trained knowledge or generating potentially inaccurate responses. Understanding the full RAG pipeline is the first step to building robust and reliable LLM applications.

At its core, the RAG pipeline involves two main phases: **indexing** your data and **generating** a response to a query. The indexing phase prepares your documents for efficient retrieval. The generation phase uses a user's query to find relevant information and then feeds that information to an LLM.

Here is a simplified view of the RAG pipeline:

```
Documents -> Chunk -> Embed -> Index (vector DB)  
Query -> Embed -> Retrieve top-k -> (Re-rank) -> Stuff into prompt -> Generate
```

Let's break down the key components and the five levers that significantly influence the quality of your RAG system.

The Five Levers of RAG Quality

1. **Chunking:** This refers to how you divide your raw documents into smaller, manageable pieces, or "chunks." The size and overlap of these chunks are crucial. If chunks are too large, they can become noisy, containing too much irrelevant information that dilutes the core context. If chunks are too small, critical context might be split across multiple pieces, making it difficult to retrieve a complete answer. A good starting point is around 500 to 800 tokens per chunk, with an overlap of about 15% between consecutive chunks to maintain context across boundaries. For example, if you have a long

technical manual, effective chunking ensures that specific procedures or definitions are contained within individual chunks.

2. **Embeddings:** Embeddings are numerical representations of text, converting words, phrases, or chunks into vectors in a high-dimensional space. These vectors capture the semantic meaning of the text. The choice of embedding model is highly important. For RAG accuracy, the quality of the embedding model used for retrieval often matters more than the specific large language model used for generation. A good embedding model will place semantically similar pieces of text closer together in the vector space.
3. **Retrieval:** After a user's query is embedded, the system searches the indexed vector database to find the most similar document chunks. This step typically involves retrieving the "top-k" most relevant chunks. Enhancements to basic vector search include **hybrid search**, which combines keyword-based search (like BM25) with vector search to capture both exact term matches and semantic similarity. **Metadata filtering** can also refine retrieval, allowing you to narrow down results based on document attributes like author, date, or topic. For instance, you might retrieve only documents published after a certain date.
4. **Re-ranking:** This is often one of the single biggest wins for RAG quality. After an initial retrieval of, say, 10 to 20 candidate chunks, a specialized model called a **cross-encoder** reorders these candidates. A cross-encoder takes both the query and each candidate chunk as input and scores their relevance more precisely than the initial embedding similarity. This ensures that the most relevant chunks are placed at the very top of the list, even if their initial embedding similarity was slightly lower than other less relevant chunks.
5. **Prompting:** The final step before generation involves constructing the prompt for the LLM. This prompt includes the user's original query and the retrieved, re-ranked context chunks. Effective prompting instructs the LLM to answer *only* from the provided context. It also explicitly tells the model to state "I don't know" or a similar phrase if the answer cannot be found within the given information. This instruction helps reduce hallucinations and ensures the LLM stays grounded in your data. For example, a prompt might begin with, "Using only the following context, answer the question. If the answer is not in the context, say 'I cannot find the answer in the provided information.'"

Debugging RAG Failures

Covers: Why RAG Fails (and Fixes)

Many issues that appear to be LLM "hallucinations" in a RAG system are actually **retrieval bugs**. This means the right information, the crucial chunk needed to answer the query accurately, never made it into the LLM's prompt. Identifying and fixing these retrieval failures is key to building a reliable RAG application.

Common Retrieval Failures and Their Fixes

1. Bad Chunking:

- **Problem:** An answer or a critical piece of context is inadvertently split across two or more chunks. When the retriever fetches only one of these partial chunks, the LLM receives incomplete information, leading to an inaccurate or incomplete answer.
- **Fixes:** Consider using larger chunks to ensure complete ideas are contained within a single piece. Alternatively, implement **sentence-aware splitting**, which attempts to keep full sentences or paragraphs together, preventing awkward breaks that fragment meaning. For example, if a definition of a term is spread across two chunks, neither chunk alone provides enough context.

2. Embedding Mismatch:

- **Problem:** The phrasing of the user's query differs significantly from the phrasing used in your source documents, even if they refer to the same concept. The embedding model may not recognize the semantic similarity, causing the retriever to miss relevant chunks.
- **Fixes:** Employ **hybrid search**, which combines traditional keyword search (like BM25) with vector search. Keyword search can catch exact term matches that vector search might miss due to phrasing differences. Another powerful fix is **query rewriting**, where an LLM is used to rephrase the user's original query into several alternative forms that might better match the document's language. For instance, if a user asks "What's the best way to clean a widget?" but your documents use "optimal cleaning procedure for device X," query rewriting can bridge that gap.

3. No Re-ranking:

- **Problem:** The initial retrieval step might fetch many candidate chunks, and the truly relevant chunk could be present but ranked lower, say at position 8. If you only pass the top 3 or 5 chunks to the LLM, the most important information is missed.
- **Fixes:** Retrieve a larger set of initial candidates, for example, 20 chunks. Then, use a dedicated re-ranker (a cross-encoder model) to reorder these candidates based on their precise relevance to the query. Finally, select the top 3 to 5 chunks from this re-ranked list to pass to the LLM. This significantly increases the chance of including the most relevant information.

4. Lost in the Middle:

- **Problem:** Large language models, particularly with longer prompts, sometimes exhibit a "lost in the middle" phenomenon. They tend to pay less attention to information buried in the middle of a long sequence of context chunks, focusing more on the beginning and end.
- **Fixes:** Strategically place the most relevant chunks at the beginning or the very end of the prompt that you send to the LLM. This ensures they receive maximum attention from the model. You might also experiment with different prompt structures to see which arrangement yields the best results for your specific LLM and task.

Essential Tools for RAG Development

To build and refine your RAG systems, several categories of tools are indispensable:

- **Vector Databases:** These specialized databases store your document embeddings and facilitate efficient similarity search. Popular options include pgvector (an extension for PostgreSQL), Qdrant, Weaviate, and Pinecone.

- **Re-rankers:** These models are crucial for improving retrieval precision. Examples include Cohere Rerank, bge-reranker, and Voyage.
- **Evaluation Frameworks:** Tools like RAGAS (available at <https://docs.ragas.io/>) help you quantitatively measure the quality of your RAG system. RAGAS provides metrics for **faithfulness** (how factual the generated answer is, based on the retrieved context) and **answer relevance** (how relevant the generated answer is to the original question), among others. Regular evaluation helps you iterate and improve your RAG pipeline systematically.

KEY TAKEAWAYS

- Retrieval-Augmented Generation (RAG) grounds LLMs in custom data, ensuring factual answers from your specific information.
- The RAG pipeline involves indexing documents (chunk, embed, index) and generating responses (query, embed, retrieve, re-rank, prompt, generate).
- Five levers control RAG quality: chunking, embeddings, retrieval strategy, re-ranking, and effective prompting.
- Most apparent LLM "hallucinations" in RAG are actually failures in the retrieval stage, meaning the correct context was not provided.
- Common retrieval failures include bad chunking, embedding mismatch between query and documents, and the absence of a re-ranking step.
- Re-ranking, using a cross-encoder to reorder retrieved chunks, is often the most significant improvement for RAG accuracy.
- Strategic placement of relevant chunks in the prompt can mitigate the "lost in the middle" problem where LLMs ignore central context.

CHECK YOUR UNDERSTANDING

Answer these before moving on. Model answers are in the Answer Key at the back of this guide.

1. What are the main stages of the RAG pipeline?
2. Name and briefly describe three of the five levers that influence RAG quality.
3. What is a common misconception about LLM "hallucinations" in a RAG system, and what is the typical underlying cause?
4. Describe one common RAG failure related to chunking and its corresponding fix.
5. How can re-ranking improve retrieval quality, and what is a practical approach to implement it?
6. What does "Lost in the middle" refer to, and how can it be mitigated?

CHAPTER 4

Final Assessment: RAG

This chapter serves as the final checkpoint for the "RAG From Scratch" course. It provides an opportunity to consolidate your understanding of Retrieval-Augmented Generation. The assessment confirms your grasp of the entire RAG pipeline, from initial document processing to generating refined LLM responses.

Use this assessment to identify your strengths and pinpoint any areas requiring further review. A thorough understanding of these concepts is essential for building effective and reliable RAG applications.

Approaching the Final Assessment: RAG

Covers: Final Assessment: RAG

The "Final Assessment: RAG" is a comprehensive quiz designed to test your mastery of the entire RAG pipeline. This includes the core stages of chunking, embedding, indexing, retrieval, re-ranking, and the critical goal of reducing LLM hallucinations. The quiz is a tool for self-evaluation, not just a pass/fail gate. It helps you confirm what you have learned and where you might need more study.

Preparing for the Assessment

Before attempting the quiz, take time to review all previous modules. Focus on the stated learning outcomes for the course:

- Understand the full RAG pipeline end to end.
- Chunk, embed, and index documents effectively.
- Build retrieval and re-ranking that actually works.
- Reduce hallucinations in LLM applications.

Revisit any lessons or concepts that felt challenging. Consider reviewing your notes or re-watching short segments of the course videos for clarity.

Strategies for Taking the Quiz

Approach each question thoughtfully. Read every question carefully to ensure you understand what it is asking. Pay attention to keywords and specific conditions mentioned. Evaluate all answer choices before selecting the best one. Do not rush through the questions; a few extra seconds of consideration can prevent simple errors. If a question seems difficult, try to eliminate obviously incorrect answers first.

Common Mistakes to Avoid

Students sometimes make common errors on assessments like this. One frequent mistake is misinterpreting the specific role of a RAG component, such as confusing the purpose of embedding with

that of indexing. Another is failing to consider the nuances of question wording, which can lead to selecting a partially correct answer instead of the most accurate one. Rushing through the choices without full consideration is also a common pitfall.

Interpreting Your Results

Your quiz results provide valuable feedback. A high score indicates a strong understanding of RAG principles and their application. If your score is lower than expected, do not be discouraged. Instead, view it as a diagnostic tool. The results highlight specific topics where your knowledge may need reinforcement.

Identify the questions you answered incorrectly. Go back to the relevant module sections and review the material. Understanding *why* an answer was wrong is more beneficial than simply knowing the correct answer. This targeted review strengthens your foundation and prepares you for real-world RAG implementation challenges.

KEY TAKEAWAYS

- The final assessment evaluates your understanding of the complete RAG pipeline.
- Effective chunking, embedding, and indexing are foundational to RAG performance.
- Successful retrieval and re-ranking directly impact the quality of LLM responses.
- Reducing hallucinations is a primary goal of well-designed RAG systems.
- Use the assessment results to identify specific topics for further study.
- A strong grasp of RAG fundamentals prepares you for building robust LLM applications.

CHECK YOUR UNDERSTANDING

Answer these before moving on. Model answers are in the Answer Key at the back of this guide.

1. What is the primary purpose of the RAG final assessment?
2. Why is effective document chunking important in a RAG pipeline?
3. How do embeddings contribute to the retrieval process in RAG?
4. What role does re-ranking play after initial document retrieval?
5. How can you use your results from the final assessment to improve your understanding?
6. What is a "hallucination" in the context of LLMs, and how does RAG aim to mitigate it?

REFERENCE

Glossary

Chunking

Breaking down large documents into smaller, manageable text segments suitable for embedding and retrieval.

Colbert (contextualized late interaction over bert)

A dense retrieval model that performs token-level similarity comparisons between query and document embeddings, rather than a single vector comparison.

Embedding

A numerical representation (vector) of text that captures its semantic meaning, allowing for similarity comparisons.

Embeddings

Numerical vector representations of text that capture semantic meaning, used to quantify the similarity between a query and document chunks.

Hallucination

In the context of LLMs, a generated response that is factually incorrect or inconsistent with the provided context, often presented confidently by the model.

Hybrid search

A retrieval strategy that combines multiple search methods, such as keyword-based search (e.g., BM25) and vector similarity search, to improve recall and precision.

Hyde (hypothetical document embedding)

A technique where an LLM generates a hypothetical answer to a query, and this generated text's embedding is used for retrieval.

Indexing

The process of preparing raw documents for efficient retrieval, involving chunking, embedding, and storing in a vector store.

Multi-representation indexing

Storing multiple versions or aspects of a document (e.g., original text, summary, Q&A pairs) in an index to improve retrieval.

Query routing

The process of directing a user's query to the most appropriate data source, tool, or sub-system based on its intent.

Query structuring

Converting a natural language query into a formal, executable language like SQL or an API call for structured data retrieval.

Query translation

Techniques that transform or expand an original user query into one or more improved queries to enhance retrieval performance.

Rag (retrieval-augmented generation)

A technique that combines information retrieval with large language models to generate responses grounded in specific, external data sources.

Raptor (recursive abstractive processing for tree-organized retrieval)

An indexing strategy that creates a hierarchical tree of document summaries to enable multi-granular retrieval.

Re-ranking

A post-retrieval step where a specialized model (often a cross-encoder) reorders an initial set of retrieved document chunks to place the most relevant ones at the top.

Reciprocal rank fusion (rrf)

A re-ranking algorithm used in RAG Fusion to combine and score documents retrieved from multiple queries, prioritizing those appearing high in several result lists.

Retrieval

The step in RAG where relevant document chunks are found based on a user's query.

Retrieval-augmented generation (rag)

A technique that combines a retrieval system with a large language model to enhance factual accuracy and relevance by providing external, specific context.

Vector database

A specialized database designed to store and efficiently query high-dimensional vectors, typically used for storing text embeddings in RAG systems.

Vector store

A specialized database designed to store and efficiently search for vector embeddings based on similarity.

REFERENCE

Answer Key

Chapter 1: Understanding Retrieval-Augmented Generation

1. Describe the three main stages of the core RAG pipeline and their purpose.

The three stages are indexing, retrieval, and generation. Indexing prepares external documents by converting them into searchable vector embeddings. Retrieval finds the most relevant document chunks based on a user's query. Generation then uses these retrieved chunks as context to help a large language model formulate an accurate answer.

2. Explain the role of chunking and embedding in the indexing stage. Why are both necessary?

Chunking breaks large documents into smaller, digestible segments, preventing context window overflow and improving search granularity. Embedding converts these text chunks into numerical vectors that capture semantic meaning. Both are necessary because chunking creates units of information, and embedding makes these units comparable and searchable in a vector space.

3. How does Multi-Query differ from a single-query retrieval, and what problem does it aim to solve?

Multi-Query generates several rephrased versions of the original user question, performing separate retrievals for each. A single-query retrieval uses only the original question. Multi-Query aims to solve the problem of ambiguous or narrowly phrased queries that might miss relevant documents, by casting a wider net and exploring multiple semantic angles.

4. What is Reciprocal Rank Fusion (RRF) and how is it used in RAG Fusion?

Reciprocal Rank Fusion (RRF) is a re-ranking algorithm that assigns a score to documents based on their positions across multiple ranked lists. In RAG Fusion, after multiple queries retrieve documents, RRF combines these results by giving higher scores to documents that appear high in several individual result sets. This helps identify the most consistently relevant documents.

5. When would you use Decomposition as a query translation technique? Provide an example.

Decomposition is used when a user's query is complex and contains multiple distinct sub-questions. It breaks the original query into simpler, atomic questions to ensure all parts are addressed. For example, if a user asks, "What is the capital of Canada and its population?", decomposition would create two sub-queries: "What is the capital of Canada?" and "What is the population of Canada?".

Chapter 2: Advanced Retrieval Techniques

1. Explain the core idea behind Hypothetical Document Embedding (HyDE) and why it can improve retrieval performance.

HyDE involves using an LLM to generate a hypothetical, plausible answer or document based on the user's original query. The embedding of this longer, generated document is then used for vector search.

This improves retrieval because the hypothetical document is often semantically richer and more similar to actual relevant documents in the index than the short, original query.

2. Describe a scenario where Query Routing would be beneficial in a RAG system.

Query routing is beneficial in systems with diverse data sources. For instance, a RAG system might have separate indices for internal company policies and external market research reports. A query about "employee leave policy" would be routed to the internal policies index, while a query about "Q3 market trends" would go to the market research index. This ensures the LLM receives context from the most relevant source.

3. How does Query Structuring differ from traditional semantic search in a RAG pipeline?

Query structuring transforms a natural language query into an executable structured query, like SQL, allowing direct interaction with databases for precise, factual answers. Traditional semantic search, conversely, embeds the natural language query and compares its vector to document embeddings in a vector store, retrieving semantically similar text chunks. Structuring is for precise data extraction, while semantic search is for finding relevant unstructured text.

4. What is the main advantage of Multi-Representation Indexing, and provide two examples of different representations.

The main advantage of multi-representation indexing is its ability to capture various facets of a document, improving retrieval flexibility and accuracy. Two examples of different representations include indexing the original document chunk alongside a concise summary of that chunk, or indexing the original chunk along with several generated question-answer pairs derived from it.

5. Briefly explain how RAPTOR helps in handling long documents more effectively than simple chunking.

RAPTOR handles long documents by creating a hierarchical tree structure. It clusters small chunks, summarizes these clusters, and then recursively summarizes the summaries. This allows the system to retrieve information at multiple granularities, from high-level topics to specific details, preserving context that simple, flat chunking often loses.

6. What is "late interaction" in the context of ColBERT, and why is it important for retrieval?

"Late interaction" in ColBERT refers to performing token-level similarity comparisons between query token embeddings and document token embeddings, rather than comparing single, aggregated vectors for the entire query and document. This is important because it allows for more granular and precise matching, capturing both semantic meaning and specific keyword relevance, leading to more accurate retrieval results.

Chapter 3: RAG Reference and Pitfalls

1. What are the main stages of the RAG pipeline?

The RAG pipeline consists of two primary stages. First, the indexing stage processes documents by chunking, embedding, and storing them in a vector database. Second, the generation stage takes a user query, embeds it, retrieves relevant chunks, optionally re-ranks them, and then uses these chunks to prompt an LLM for a response.

2. Name and briefly describe three of the five levers that influence RAG quality.

Three levers influencing RAG quality are chunking, embeddings, and re-ranking. Chunking involves dividing documents into optimal sizes with appropriate overlap to preserve context. Embeddings refer to the numerical representations of text; the quality of the embedding model is crucial for accurate similarity search. Re-ranking is the process of reordering initially retrieved chunks using a specialized model to ensure the most relevant information is prioritized for the LLM.

3. What is a common misconception about LLM "hallucinations" in a RAG system, and what is the typical underlying cause?

A common misconception is that LLM "hallucinations" are solely due to the LLM itself making up facts. In a RAG system, most such issues are actually retrieval bugs. This means the correct, factual information was available in the source documents but was not successfully retrieved and passed to the LLM in the prompt.

4. Describe one common RAG failure related to chunking and its corresponding fix.

A common chunking failure occurs when a complete answer or critical piece of context is split across two or more separate chunks. This leads to the LLM receiving only partial information. A fix involves using larger chunk sizes to encompass full ideas or implementing sentence-aware splitting to prevent breaking sentences or paragraphs.

5. How can re-ranking improve retrieval quality, and what is a practical approach to implement it?

Re-ranking improves retrieval quality by more accurately identifying and prioritizing the most relevant chunks from an initial set of candidates. It uses a cross-encoder model to score query-chunk pairs, providing a more refined relevance assessment than initial embedding similarity. A practical approach is to retrieve a larger number of initial chunks (e.g., 20), then pass them through a re-ranker, and finally select the top 3-5 re-ranked chunks for the LLM prompt.

6. What does "Lost in the middle" refer to, and how can it be mitigated?

"Lost in the middle" describes the phenomenon where LLMs tend to pay less attention to context chunks located in the middle of a long prompt, focusing more on information at the beginning or end. This can be mitigated by strategically placing the most relevant retrieved chunks at the very beginning or the very end of the prompt provided to the LLM, ensuring they are more likely to be utilized.

Chapter 4: Final Assessment: RAG**1. What is the primary purpose of the RAG final assessment?**

The primary purpose is to confirm your understanding of the entire RAG pipeline and its components. It serves as a self-evaluation tool to highlight areas of strength and topics that may require further review.

2. Why is effective document chunking important in a RAG pipeline?

Effective chunking breaks down large documents into manageable pieces. This improves the relevance of retrieved information, prevents context window overflow, and ensures that embeddings accurately represent specific ideas, leading to better retrieval results.

3. How do embeddings contribute to the retrieval process in RAG?

Embeddings convert text into numerical vectors that capture semantic meaning. During retrieval, the query's embedding is compared to document chunk embeddings to find semantically similar pieces of information, enabling efficient and relevant searches.

4. What role does re-ranking play after initial document retrieval?

Re-ranking refines the initial set of retrieved documents by applying a more sophisticated scoring mechanism. This process helps to prioritize the most relevant chunks, ensuring that the LLM receives the highest quality context to generate its response.

5. How can you use your results from the final assessment to improve your understanding?

You should use your assessment results diagnostically. Identify incorrect answers and review the corresponding course material. Understanding the reasons behind your mistakes is key to reinforcing your knowledge and improving your grasp of RAG concepts.

6. What is a "hallucination" in the context of LLMs, and how does RAG aim to mitigate it?

An LLM hallucination occurs when the model generates factually incorrect or unsupported information. RAG mitigates this by providing the LLM with relevant, verified information from an external knowledge base, grounding its responses in factual context rather than relying solely on its training data.

CLOSING

Where to Go Next

After completing "RAG From Scratch", consolidate your skills by building your own Retrieval-Augmented Generation applications. Start with a simple project, such as a Q&A system for a small document set, and gradually increase complexity. Experiment with different chunking strategies, embedding models, and re-ranking algorithms to see their impact.

Consider integrating RAG into an existing LLM project or developing a new application that benefits from grounded responses. Practical application is key to mastering these techniques.

To continue your learning, explore advanced RAG architectures, multi-modal RAG, or the integration of RAG with agentic workflows. Understanding how RAG fits into broader AI systems will further enhance your capabilities.



MEGACOURSE

Learn Anything. Achieve Everything.

Track your progress, take the quizzes, and earn your certificate for this course at
megacourse.app/courses/rag-from-scratch