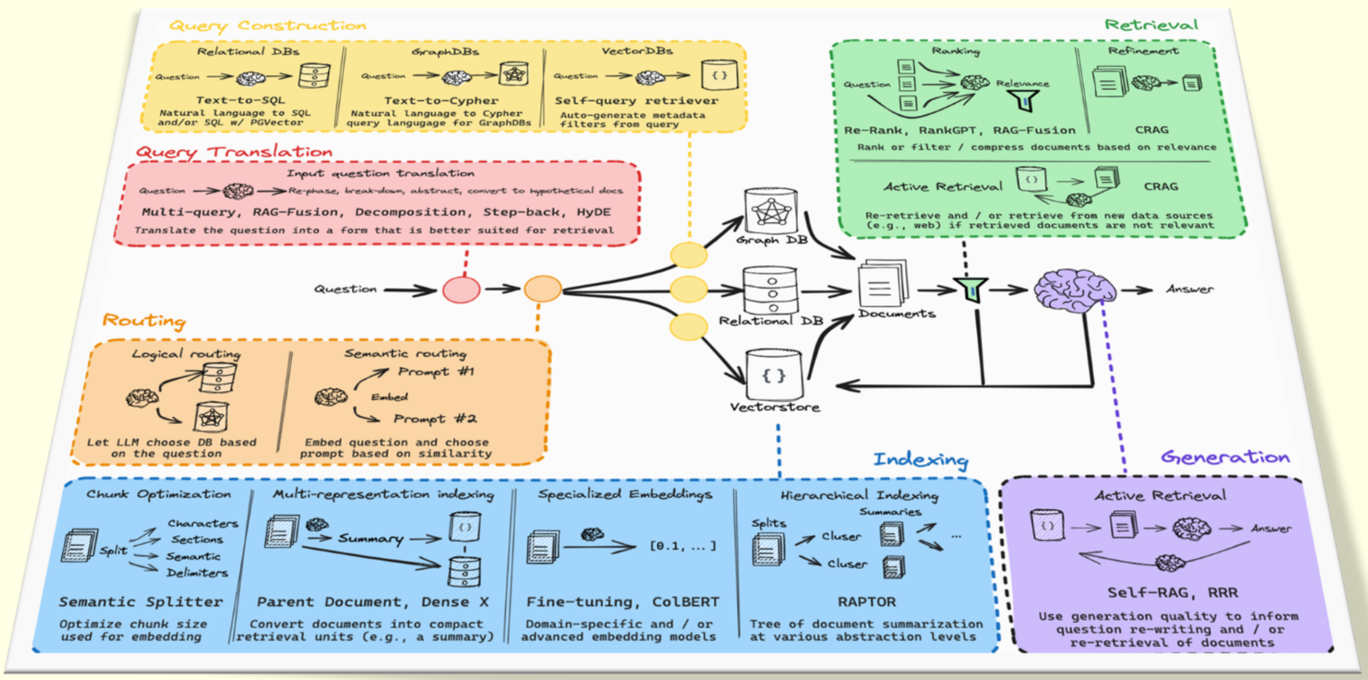
RAG Tutorial

(Retrieval Augmented Generation)

Retrieval and Generation: Retrieve


Query Construction
Relational DBs
Question
Text-to-SQL
Natural language to SQL and/or SQL w/ PGVector
GraphDBs
Text-to-Cypher
Natural language to Cypher query language for GraphDBs
VectorDBs
Self-query retriever
Auto-generate metadata filters from query
Query Translation
Input question translation
Re-phase, break-down, abstract, convert to hypothetical docs
Multi-query, RAG-Fusion, Decomposition, Step-back, HyDE
Translate the question into a form that is better suited for retrieval
Retrieval
Ranking
Relevance
Re-Rank, RankGPT, RAG-Fusion
Refinement
CRAG
Rank or filter / compress documents based on relevance
Active Retrieval
CRAG
Re-retrieve and / or retrieve from new data sources (e.g., web) if retrieved documents are not relevant
Graph DB
Relational DB
Vectorstore
Documents
Answer
Routing
Logical routing
Let LLM choose DB based on the question
Semantic routing
Prompt #1
Embed
Prompt #2
Embed question and choose prompt based on similarity
Indexing
Chunk Optimization
Split
Characters
Sections
Semantic
Delimiters
Semantic Splitter
Optimize chunk size used for embedding
Multi-representation indexing
Summary
Parent Document, Dense X
Convert documents into compact retrieval units (e.g., a summary)
Specialized Embeddings
[0.1, ...]
Fine-tuning, ColBERT
Domain-specific and / or advanced embedding models
Hierarchical Indexing
Splits
Summaries
Cluser
Cluser
RAPTOR
Tree of document summarization at various abstraction levels
Generation
Active Retrieval
Answer
Self-RAG, RRR
Use generation quality to inform question re-writing and / or re-retrieval of documents

Day 6 of 7

Table of Contents

1. Introduction

In the previous lesson, we explored how to embed and store document splits in a vector store using LangChain.

Today, we'll focus on the retrieval process—specifically, how to create a simple application that takes a user query, searches for relevant documents, and returns an answer.

This is a crucial step in building a Retrieval-Augmented Generation (RAG) system, where the goal is to augment the model's generation capabilities with retrieved information from relevant documents.

2. Defining the Retrieval Logic

- The retrieval process is at the heart of any RAG system. LangChain provides a robust interface called Retriever that simplifies this process.
- The Retriever interface wraps an index and returns relevant documents given a string query.
- This interface abstracts away the complexity of searching and retrieving, allowing you to focus on higher-level application logic.

- **The most commonly used retriever in LangChain is the VectorStoreRetriever, which leverages the similarity search capabilities of a vector store to find relevant documents.**

- **You can easily convert any vector store into a retriever using the VectorStore.as_retriever() method.**

3. Setting Up the VectorStoreRetriever

The first step in retrieval is configuring the VectorStoreRetriever. This object will allow us to perform a similarity search on the vector store to find documents that are most relevant to the user's query.

Here's how you can set it up:

```
# Assume vectorstore is already created and contains
document embeddings
retriever = vectorstore.as_retriever(
                                search_type="similarity",
                                search_kwargs={"k": 6})
```

- **search_type="similarity":** This specifies that the retrieval method will be based on similarity search.

- **search_kwargs={"k": 6}:** This parameter determines the number of documents to retrieve. In this case, we want the top 6 most relevant documents.

4. Retrieving Documents

Indexing With the VectorStoreRetriever set up, retrieving documents is straightforward. You simply pass in a query string, and the retriever will return the most relevant documents.

```
retrieved_docs = retriever.invoke("What are the approaches
                                   to Task Decomposition?")
```

Here's what happens under the hood:

- **The query "What are the approaches to Task Decomposition?" is transformed into an embedding vector.**
- **The retriever then compares this vector against the vectors stored in the vector store.**
- **The top 6 most similar documents are returned as retrieved_docs.**

To verify the retrieval process, you can check the number of documents retrieved:

```
len(retrieved_docs)  # Should return 6
```

And to inspect the content of the first retrieved document:

```
print(retrieved_docs[0].page_content)
```

This might output something like:

```
Tree of Thoughts (Yao et al. 2023) extends CoT by exploring
multiple reasoning possibilities at each step...
```

This snippet shows that the retriever successfully fetched a document relevant to the topic of task decomposition.

5. Exploring Other Retrieval Methods

While vector stores are the most common approach for document retrieval, LangChain supports several other retrieval techniques. Here are a few worth exploring:

- **MultiQueryRetriever:** Enhances retrieval accuracy by generating multiple variations of the input query.
-
- **MultiVectorRetriever:** Improves retrieval by generating multiple variations of embeddings.

- **Max Marginal Relevance (MMR):** Prioritizes diversity in the retrieved documents, avoiding duplicates and ensuring a richer context.

- **Self Query Retriever:** Allows filtering of documents during retrieval using metadata filters.

These advanced techniques offer greater flexibility and precision in retrieval, especially in complex applications.

6. Sample Code and Explanation

Define the Text Chunks

Instead of defining a pre-set list, we'll demonstrate how to split a larger document into smaller, more manageable text chunks. This process is crucial for efficient retrieval and will be explored in detail

Embed and Store Document Splits

As covered in Day 5, these chunks are converted into vectors and stored in a vector store.

Querying the Vector Store

The core of today's lesson—using a retriever to find relevant documents based on a user query.

Inspect the Vector Store

You can view the retrieved documents to confirm that the retrieval process is working correctly

Link of collab Notebook

7. Conclusion

Today's lesson highlighted the critical role of the Retriever in a RAG pipeline. By setting up a VectorStoreRetriever, we can efficiently find documents that are most relevant to a user's query.

Understanding and implementing retrieval techniques is essential in building applications that can provide accurate and contextually rich responses to user queries.

Stay tuned for Day 7, we'll dive into the generation phase, where we'll combine the retrieved documents with the initial query to generate a final answer