Retrievers in LangChain

1. Introduction to LangChain and RAG

Retrievers as a crucial component for building **RAG (Retrieval-Augmented Generation) based applications**. RAG systems aim to enhance the capabilities of Language Models (LLMs) by providing them with relevant external information.

Before delving into RAG, the series covers its core components:

1. **Document Loaders**: For loading data.
2. **Text Splitters**: For breaking down large texts.
3. **Vector Stores**: For storing and retrieving vector embeddings of documents.
4. **Retrievers**: The focus of this video, responsible for fetching relevant documents.

Once these four components are understood, the audience will be ready to learn about RAG.

2. What are Retrievers?

In simple terms, "A retriever is a component in LangChain that fetches relevant documents from a data source in response to a user's query."

How it works:

- A **data source** (e.g., a Vector Store, API, etc.) stores all the data.
- A **user sends a query** to the system.
- The **Retriever** receives the query, internally accesses the data source, scans documents, and identifies the most relevant ones.
- It then **fetches these relevant documents** and provides them as output.

Key Characteristics of Retrievers in LangChain:

- **Function-like behavior**: A retriever can be thought of as a function that takes a user query as input and outputs "multiple Document objects."
- **Search Engine Analogy**: Its internal processing involves searching a data source, similar to how a search engine provides relevant results.
- **Multiple Types**: "LangChain में सिर्फ एक रिट्रीवर नहीं होता There are multiple retrievers for different use cases."
- **Runnables**: "LangChain में जितने भी रिट्रीवर्स आपको देखने को मिलेंगे वो सब के सब runnables हैं Just like models, prompts etc." This means they can be used to form or be plugged into existing chains, significantly enhancing system flexibility.

3. Types of Retrievers

Retrievers can be categorized in two primary ways:

3.1. Based on Data Source

Different retrievers are designed to work with specific types of data sources.

- **Wikipedia Retriever**: Queries the Wikipedia API to fetch relevant articles based on a user's query. It primarily uses "keyword matching" for relevance, not semantic search.

- **Vector Store Based Retrievers**: Work with data stored in a vector store. They convert the user's query into a vector embedding and perform "semantic similarity" search against document embeddings to find relevant results.
- **Archive Retriever**: Fetches relevant research papers from the Archive website.
- Many more exist, categorized by the type of data source they interact with.

3.2. Based on Search Strategy (Retrieval Mechanism)

Different retrievers employ distinct mechanisms or algorithms for searching and retrieving documents.

- **MMR (Maximum Marginal Relevance) Retriever**: Aims to "reduce redundancy in the retrieved results while maintaining high relevance to the query." It selects documents that are not only relevant but also "different from each other," providing a diverse set of results.
- **Multi-Query Retriever**: Solves the problem of "ambiguous" user queries. It uses an LLM to "generate multiple queries" from a single ambiguous query, then sends all these generated queries to a base retriever (like a similarity search retriever). The results from all sub-queries are merged, and duplicates are removed to provide comprehensive and less ambiguous results.
- **Contextual Compression Retriever**: An "advanced retriever that improves retrieval quality by compressing documents after retrieval, keeping only the relevant content based on the user's query." It's particularly useful when documents contain "mixed information" or when text splitters might have created documents with irrelevant sections. It uses a "base retriever" to fetch initial documents and then a "compressor" (usually an LLM) to trim irrelevant parts, returning only query-relevant content.

4. Types of some Specific Retrievers

4.1. Wikipedia Retriever

- **Functionality**: Queries the Wikipedia API.
- **Parameters**: top_k (number of results) and lang (language).
- **Underlying Mechanism**: Primarily keyword matching, not semantic search.
- **Runnable**: Demonstrated by calling retriever.invoke(query).
- **Distinction from Document Loaders**: It's a retriever because it performs "searching" and "decides which documents are relevant," rather than just loading all articles. It has "some form of intelligence" in deciding relevance.

4.2. Vector Store Retriever

- **Functionality**: Fetches documents from a vector store based on semantic similarity using embeddings.
- **Setup**: Requires setting up a vector store (e.g., Chroma) with embeddings (e.g., OpenAI Embeddings).
- **Usage**: vector_store.as_retriever(k=...) creates the retriever.
- **Comparison to Direct Vector Store Search**: While a vector store can directly perform similarity_search, the Vector Store Retriever is a "runnable" and can be integrated into chains. More importantly, it serves as a "vanilla" base for implementing "advanced search strategies" through other retriever types (like MMR, Multi-Query, Contextual Compression) that leverage the vector store.

4.3. MMR (Maximum Marginal Relevance) Retriever

- **Problem Solved**: Addresses redundancy in search results where multiple highly relevant documents might convey the same information.
- **Core Philosophy**: "How can we pick results that are not only relevant to the query but also different from each other."

- **Implementation**: Created by specifying search_type="mmr" and a lambda_mult parameter (0 to 1, where 1 acts like normal similarity search, 0 gives highly diverse results).
- **Mechanism**: First picks the most relevant document, then subsequent picks are highly relevant yet highly dissimilar from previously picked documents.

4.4. Multi-Query Retriever

- **Problem Solved**: Handles "ambiguous" user queries that could have multiple interpretations.
- **Core Idea**: "It tries to resolve the ambiguity within the user's query by generating multiple queries."
- **Mechanism**:

1. User sends an ambiguous query.
2. An LLM generates "multiple, diverse, but related queries" from the original.
3. Each generated query is sent to a base retriever (e.g., similarity search retriever).
4. Results from all sub-queries are merged, and duplicates are removed.
5. Top results are shown to the user.

- **Implementation**: Requires an LLM for query generation and a base retriever for document retrieval.

4.5. Contextual Compression Retriever

- **Problem Solved**: Addresses the issue where retrieved documents contain irrelevant information alongside relevant parts, especially if original documents were large and split imperfectly.
- **Core Idea**: "Compresse documents after retrieval, keeping only the relevant content based on the user's query."
- **Mechanism**:

1. A "base retriever" fetches initial relevant documents.
2. A "compressor" (typically an LLM or LLMChainExtractor) is applied to each retrieved document.
3. The compressor "trims" the document, retaining only the parts relevant to the user's query and discarding irrelevant content.

- **Use Cases**: When documents are very long or contain mixed information, or to reduce LLM context length and improve RAG pipeline accuracy.

5. Conclusion and Further Exploration

- Many other retrievers exist in LangChain (e.g., Parent Document Retriever, Time Weighted Vector Retriever, Self-Query Retriever, Ensemble Retriever). LangChain documentation as the resource for exploring these.
- The primary reason for the existence of so many diverse retrievers is to **improve the performance of RAG-based systems**. A simple RAG system might not always yield optimal results. Advanced retrievers are key to enhancing RAG system performance by providing more refined and relevant context to the LLM.
- Understanding these retrievers is crucial for building and improving advanced RAG applications in the future.