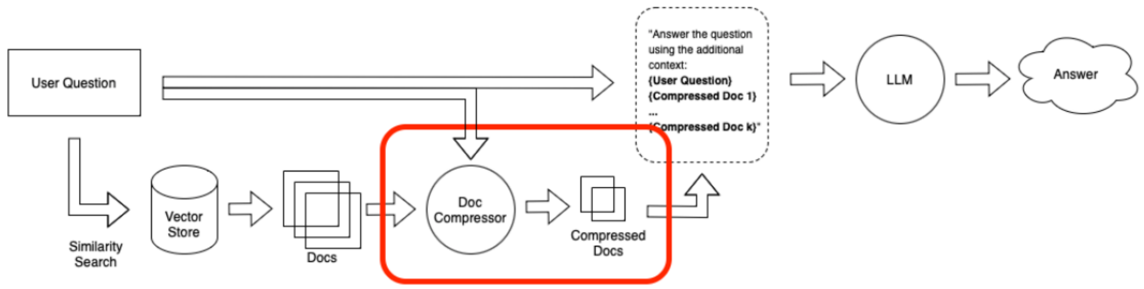
advanced RAG-tec

▼ Contextual Compression

Contextual compression aims to enhance the retrieval process by focusing on returning only the most relevant parts of documents, reducing irrelevant content and the overall number of documents. This approach can significantly improve the efficiency and quality of responses from a language model.

The goal of compressors is to make it easy to pass **only** the relevant information to the LLM. By doing this, it also enables you to pass along **more** information to the LLM, since in the initial retrieval step you can focus on recall (e.g. by increasing the number of documents returned) and let the compressors handle precision.



Retrieval Q&A system with contextual document compression

Here's how to implement contextual compression using different methods hand a combination of compressors in LangChain.

▼ vanilla vector store

Retrieving Text Chunks Using a Vector Store

1. **Initialization:**
    - Set up a vector store (e.g., FAISS, Pinecone) and a retriever to find similar vectors.
2. **Chunking the Document:**

    - Split the document into smaller chunks and encode each chunk into vectors using a pre-trained embedding model.

3. **Storing Chunks:**
    - Store the vectors in the vector store for efficient similarity searches.
4. **Retrieving Relevant Chunks:**
    - When a query is made, search for chunks whose vectors are most similar to the query vector.

Observations

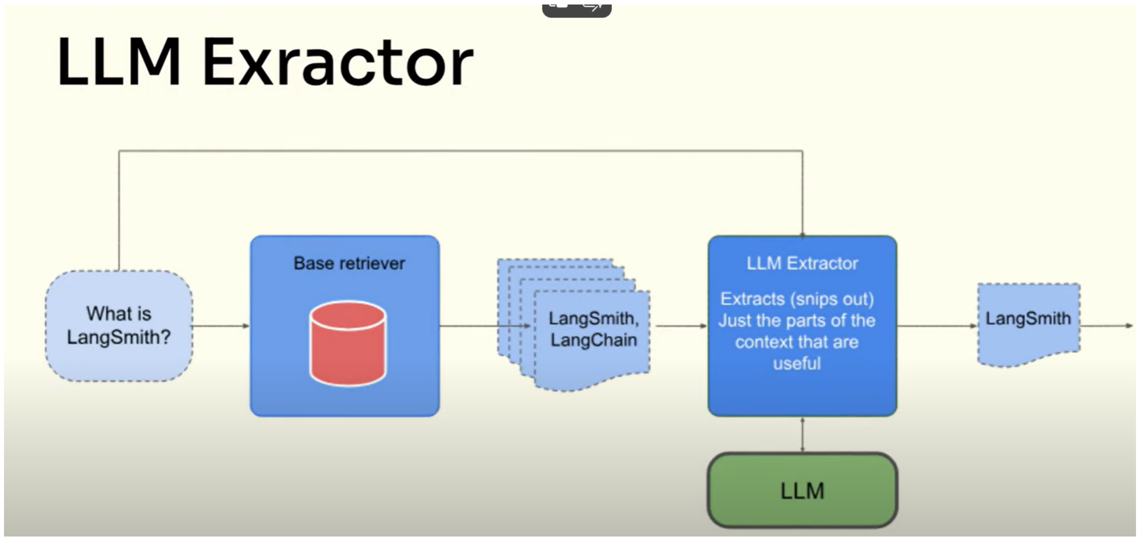
- **Relevance of Retrieved Chunks:**
    - A vanilla vector store retriever may return less relevant chunks if their vectors are close to the query vector. Some retrieved chunks might contain unrelated information despite their apparent similarity.

Conclusion

- A vector store retriever is useful for retrieving text chunks based on semantic similarity, but advanced techniques may be needed to enhance the relevance of the retrieved information.

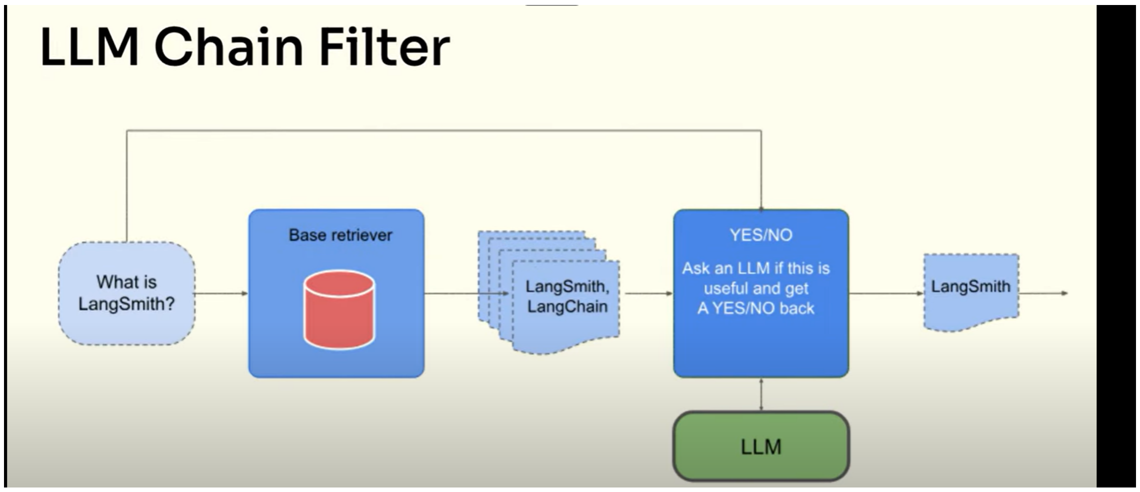
▼ LLMChainExtractor

The LLMChainExtractor extracts the most relevant content from documents based on a query. It uses a language model to identify and focus on the parts of the documents that are most relevant, summarizing the information to what's needed.



▼ LLMChainFilter

The `LLMChainFilter` is a straightforward but effective tool that uses an LLM chain to decide which retrieved documents to keep or discard, without altering the document contents.

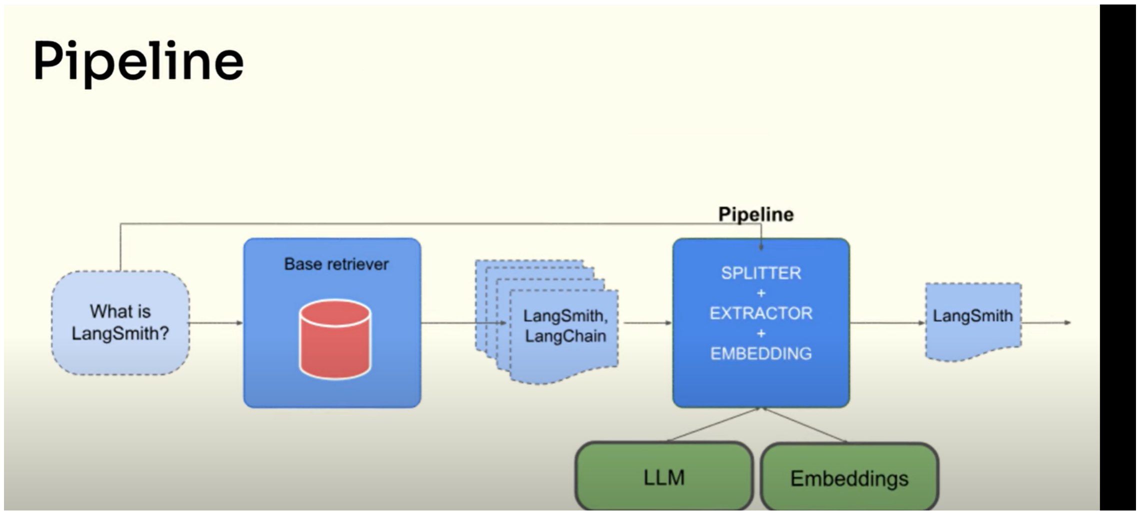


▼ EmbeddingsFilter

The `EmbeddingsFilter` is a faster and more cost-effective alternative. It embeds the documents and the query, then returns only those documents with embeddings similar enough to the query, avoiding extra LLM calls (expensive and slow)

▼ Pipeline

The DocumentCompressorPipeline allows us to combine multiple compressors in sequence for efficient document processing. Alongside compressors, we can include BaseDocumentTransformers, which transform documents without compressing them.



▼ **Ensemble Retriever**

The Ensemble Retriever method combines multiple retrieval models to improve the overall retrieval performance. By leveraging the strengths of different retrieval models, an ensemble can provide more accurate and diverse results compared to a single model. The ensemble can include a combination of traditional and neural retrieval models, each contributing to the final set of retrieved documents. The idea is to aggregate results from different models, which can complement each other and provide a more comprehensive retrieval output.

Key Points:

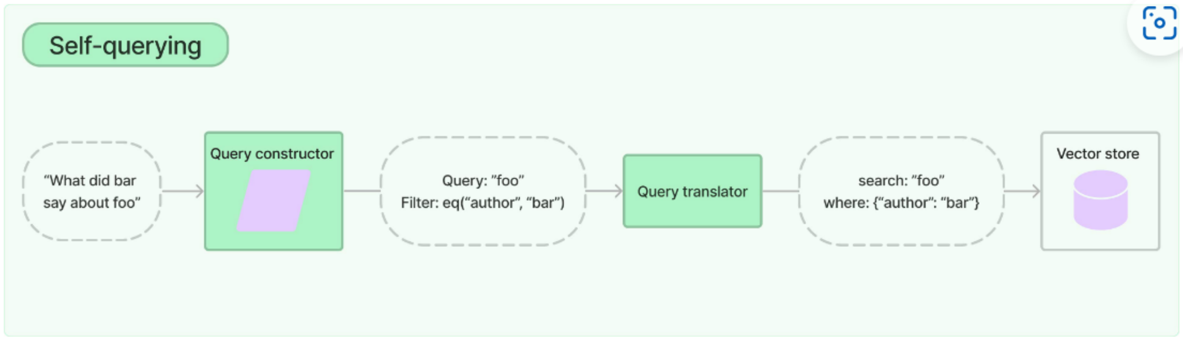
- Combines multiple retrieval models.
- Provides more accurate and diverse results.
- Aggregates results to leverage strengths of different models.

Code:

The most common pattern is to combine a sparse retriever (like BM25) with a dense retriever (like embedding similarity), because their strengths are complementary. It is also known as "hybrid search". The sparse retriever is good at finding relevant documents based on keywords, while the dense retriever is good at finding relevant documents based on semantic similarity.

▼ **Self-Querying Retriever**

A **self-querying retriever** is an advanced search tool that can process natural language queries by first converting them into structured queries using a language model. It then searches through a database (VectorStore) not just for semantic matches, but also applies filters based on metadata extracted from the query. This approach improves search accuracy and relevance by combining meaning-based search with metadata filtering.



▼ **Time-weighted Vector Store Retriever**

The **Time-Weighted Vector Store Retriever** is a mechanism used to retrieve documents based on both their semantic relevance and their recency, incorporating a time decay factor. This approach ensures that more recent documents are given higher priority, while older documents gradually become less relevant unless they are frequently accessed.

How It Works

1. **Scoring Mechanism:**
   The retriever scores documents using the following formula:Score=semantic_similarity+(1.0−decay_rate)hours_passed

   Score=semantic_similarity+(1.0−decay_rate)hours_passed\text{Score} = \text{semantic_similarity} + (1.0 - \text{decay_rate})^{\text{hours_passed}}

   - **semantic_similarity:** This represents how closely a document matches the query in terms of content.
   - **hours_passed:** The number of hours since the document was last accessed.
   - **decay_rate:** A parameter that controls how quickly the recency score decays over time.
2. **Effect of Decay Rate:**
   - **Low Decay Rate:** A low decay rate (e.g., close to 0) means that documents remain "fresh" for a longer period, making the retriever behave almost like a traditional vector search.
   - **High Decay Rate:** A high decay rate causes the recency score to decay quickly, so older documents lose relevance rapidly.
3. **Virtual Time:**
   - The retriever can mock or manipulate the time component using utilities like `mock_now`, allowing for controlled testing or simulations of time-sensitive retrieval.