Source Code: <https://github.com/langchain-ai/rag-from-scratch>

Youtube video:

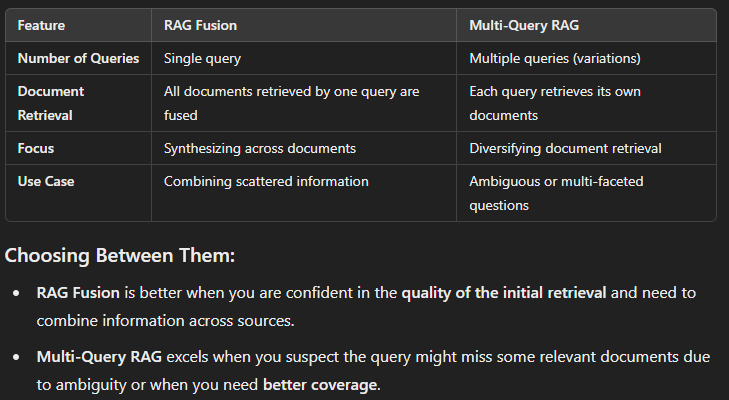
1. Documents - in PDF, word, txt, spreadsheet etc.
2. Partitioning into blocks - using customised partitioning or automated
3. Embeddings - Changing texts into vector (numeric representations) to compute relationships and entities
4. Indexing
5. Vector store - embeddings are stored in a high dimensional vector space (2D or 3D) represented by vector points located in each region of the vector space
6. User prompt input - user inputs questions in the form of a string of letters
7. Prompt embedding - text prompt represented in vector as well
8. Vector prompt store - vector prompt is stored in the same high dimensional vector space as document vectors
9. Comparison and return - compare similar vector points in the vector space to return relevant documents. Typically done by using the prompt vector and find k number of neighbour vector points (relevant documents)
10. Question (prompt) and relevant documents are passed as key value pairs into dictionaries
11. This is used to populate the prompt template (decided by user via code) and populated using relevant documents and prompt
12. Prompt template is passed through LLM
13. Return message obtained is parsed
14. Return message is printed

***Query Translation → Query modification to improve retrieval***

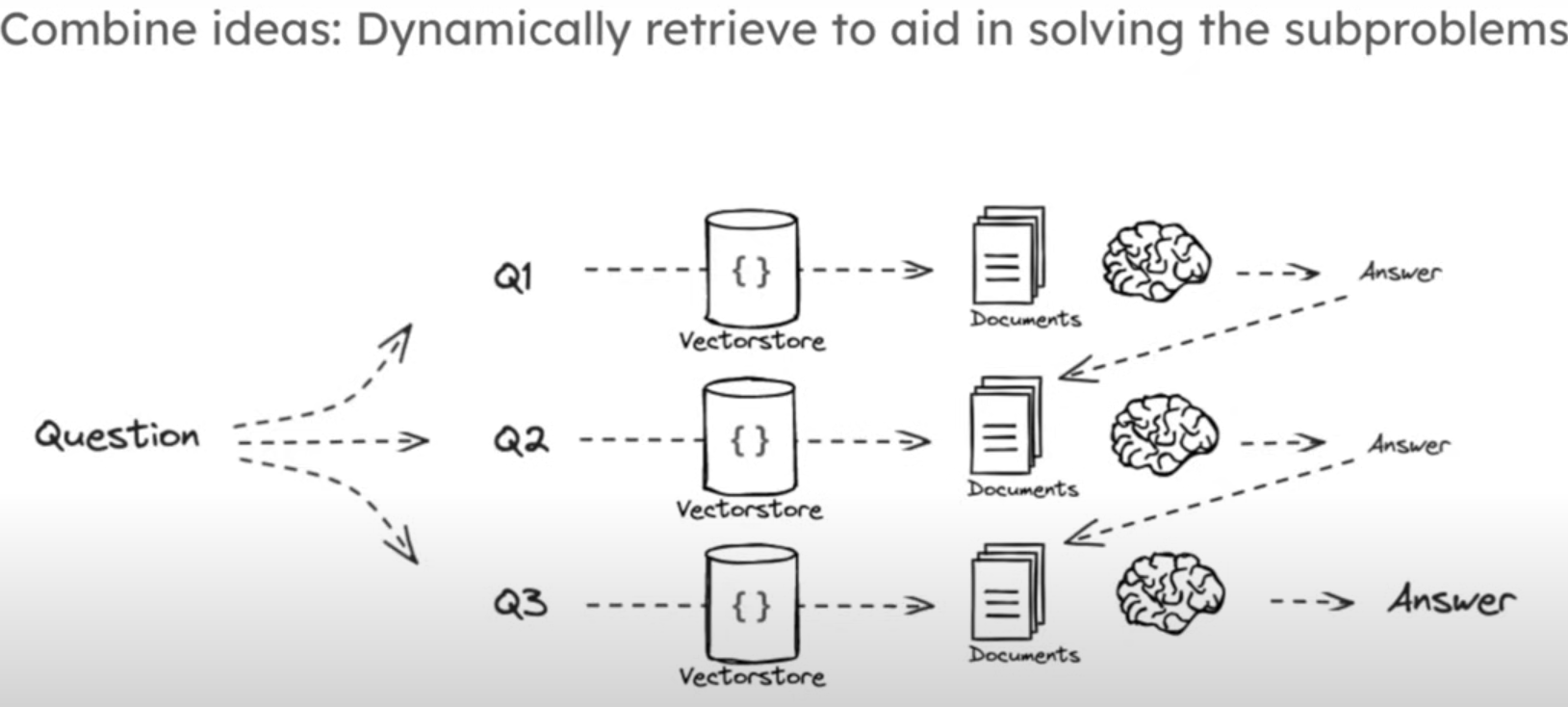
Query Translation - Multi Query

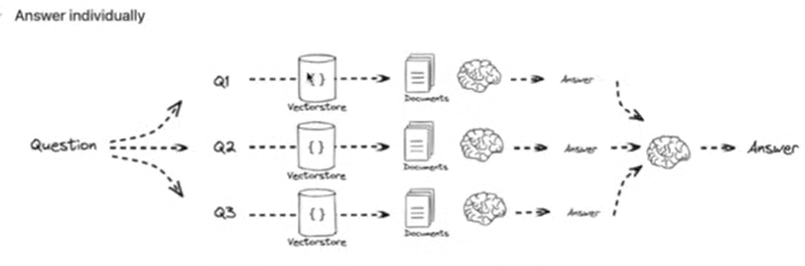
* Rewriting the input question into multiple questions in different ways, the carry out parallel retrieval to get relevant documents.

Query Translation - RAG Fusion



Query Translation - Decomposition





Query Translation - Step-back Prompting

* Asking a more generic question from the user input, and combine the retrieved documents to generate an output

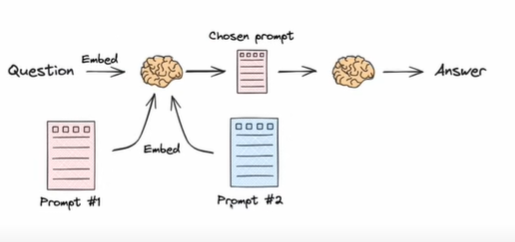
Query Translation - HyDE

Generating a hypothetical document based on the input query in hopes that a document will have higher similarity to vector indices generated by document chunks in vector space

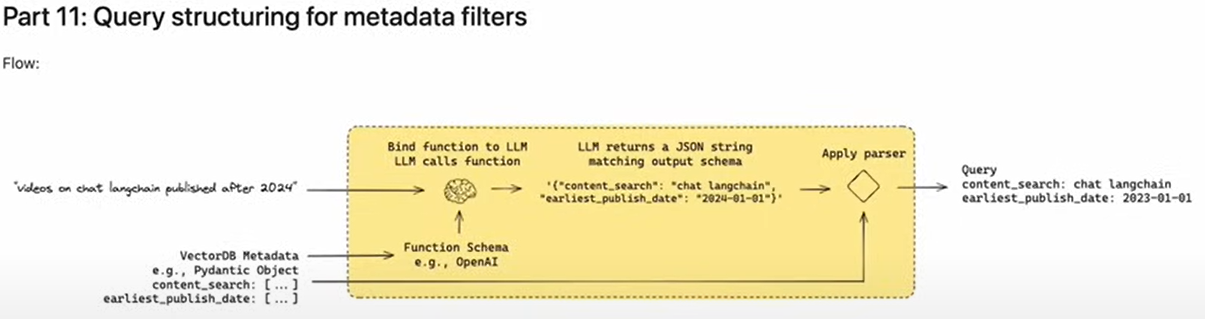
Routing

* Direct queries to appropriate retrieval or generative components
* eg) python, json, etc.

Semantic routing

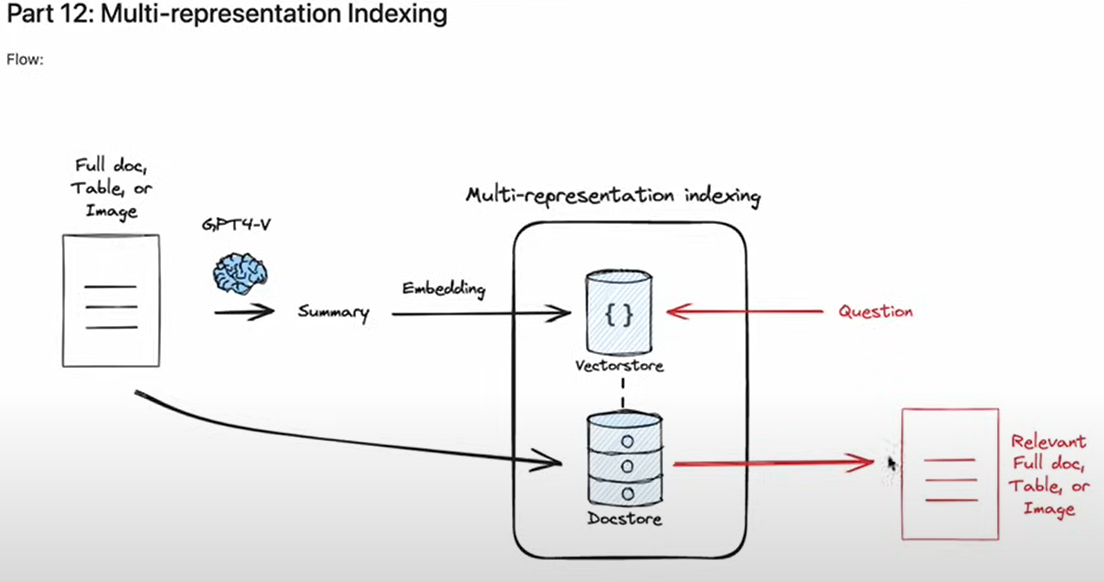


Query Construction



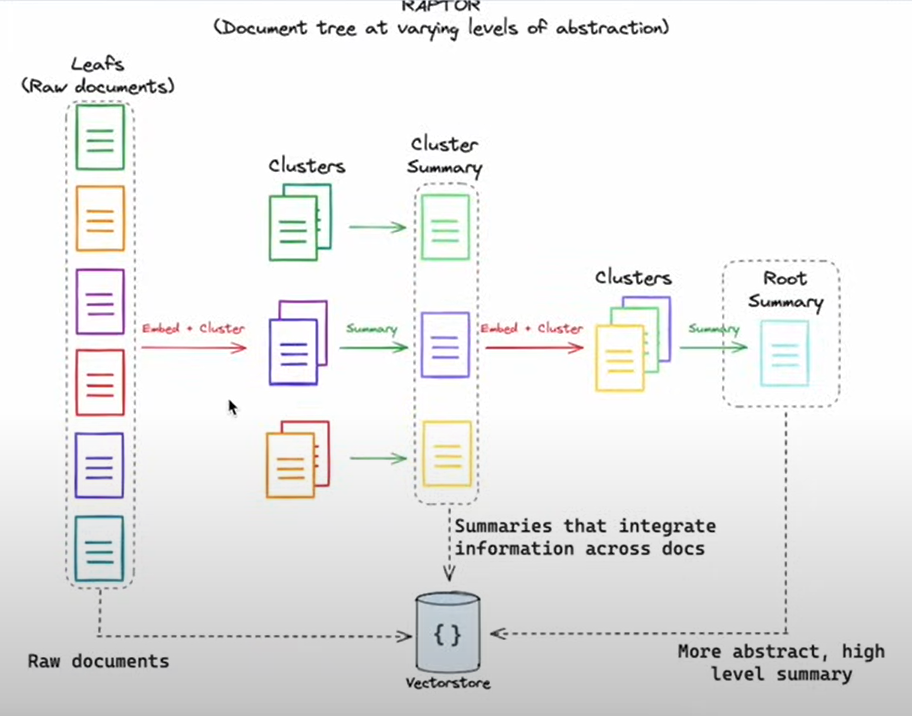
Basically applying metadata filters to retrieve relevant documents, like date published, author etc. Changing unstructured prompt input into structured metadata filters

Multi-Representation Indexing

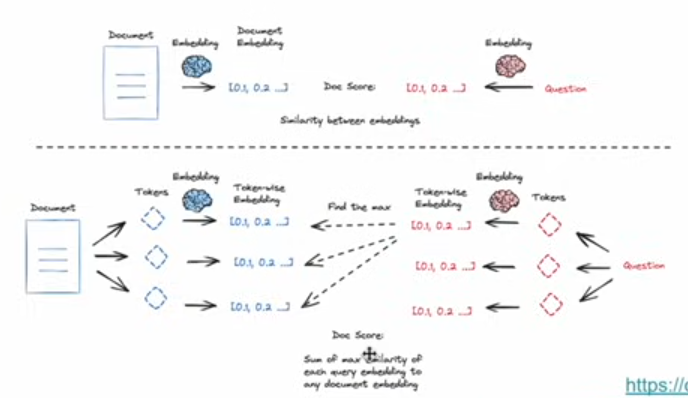


* Creating a summary of a doc to be stored in a vector store and the original document in a document store. Then applying a query prompt to find relevant data from vector store (summary) then use index to reference and retrieve the original document

RAPTOR indexing



Colbert Indexing



Adaptive RAG - Using fallback mechanism to reflect on every stage to generate more relevant and accurate output, avoiding hallucinations

