By Carson Crockett

RAG Summary and LangChain Methodologies

This document is meant to describe the general RAG pipeline and as well outline some methods which can be used in Query Translation. It is mostly based on the information from this video: https://www.youtube.com/watch?v=sVcwVQRHIc8, specifically the first hour which covers the RAG pipeline and Query Translation

RAG

Retrieval Augmented Generation consists of three main steps. Indexing, Retrieval, and Generation.

Indexing is defining the context or data to be used in generation. We take a set of documents and index them so that later during retrieval we can determine which documents are necessary/relevant. Documents are indexed according to heuristics derived from the document. For example imagine the documents being assigned a spot in a 3-D space. It doesn't need to be 3-D but it makes it easy to understand. The goal of RAG is to take an input query and map it to a spot in the same 3-D space, take the documents nearby as relevant and then pass the question to an LLM with the relevant documents as the context for the question. The LLM then generates a response based on the information it gets from the relevant documents.

Indexing

Indexing consists of two parts, splitting and embedding. Splitting is just dividing the document into pieces, this is done for convenience. Embedding is the process of mapping the document to the space. This is done by compressing the document into a vector which captures the semantic meaning of the document. The dimensions of the space are determined based on the subject matter and the documents are embedded based on their relevance to those dimensions. At this point it's a black box which is handled by pre-built implementations from LangChain but know this is what determines where each document goes where in the space.

Retrieval

So we now have a space with each document in our dataset stored in it at a certain location with a reference to the actual document to be pulled later. Now we need to determine which of those documents are related to the query the user sends us. To do this we embed the query in the same space and say that the nearby documents are relevant. Developers can state that the k nearest documents to the embedded query are relevant. For example if k = 3 we would take the 3 closest documents to where the query is mapped after embedding.

Generation

So now we have all the relevant documents we need to answer the query. With that we take the query provided by the user and send it to the LLM. We give the LLM a prompt

similar to “Answer this question based on this context” with the question being the query and the context being the relevant documents. There are more sophisticated prompts LangChain provides but they follow that general structure.

What does LangChain do

LangChain is a framework which allows this to happen. It has integrations which can do these processes for us we just need to pick which ones to use and specify them in our implementation. LangChain has integrated document loaders, indexers, document splitters, and embedders.

Query Translation

The problem we run into is that user queries aren’t always great and the embedding and generation won’t always perfectly match what the user wanted, the key is that it will get close. Below we’ll cover methods we can use to modify the query to narrow in on what exactly the user wants.

Multi-Query and RAG-Fusion / Query Re-writing

These methods focus on rewording the query in several ways and running generation on all of them. Similar to widening the scope of the problem we get all the relevant documents for each reworking of the query and use them all to generate an answer.

Multi-Query Transformation takes the query and rewords it into several different queries. We take each new query and embed it and find the relevant documents to it. From this we have a large list of relevant documents for each query. We take the entire list and pass it to the LLM as the context and the original query as the question and use it to generate an answer. This method will help make sure the proper relevant information is given to the LLM.

RAG-Fusion Transformation is similar to multi-query. It also rewords the query into multiple queries and embeds it to find relevant documentation. The difference is that when the relevant documentation from the several queries are compiled they are ranked using something called Reciprocal Rank Fusion. This process ranks all the documents based on their relevance to the original query. From here we can specify to only use the top n most relevant documents. This allows us to broaden the scope of the query to get more relevant information and then narrow it back down to avoid taking in unnecessary documents.

Decomposition

This is a different approach to Query Transformation which focus on breaking the query into it’s components rather than expanding it into multiple queries and “solving” each component. One approach to this is the Leat-to-Most approach. Here we break the query down into it’s most simple component and up until it is back to the original query. We then solve the most simple component, then for the next and subsequent components

we provide the answer to the previous component to solve the current one until we generate a solution for the original query. For example if we asked “What is a LLM, what are it’s uses, and can I use it to solve this problem” we could break it down into “what is an LLM”, answer that, use the answer to solve, “What is a LLM, What are it’s uses”, and use that answer to solve “What is a LLM, What are it’s uses, can I use it to solve this problem”. With each iteration we get closer to the original query with more background information at each step to inform the answer.

Step-Back

This method of Query Transformation is my personal favorite for the purpose of the Michelin Project. Instead of becoming less abstract like Least-to-Most Step-Back generates more abstract queries from the original query. We ask multiple abstract questions which address the necessary information to answer the original question. For example if the user asks “How do I bake a cake” we could ask “What is a cake?”, “What ingredients do I need for a cake?”, “What steps are needed to bake a cake?” and use the answers to these questions to answer the original query.

HyDE

This method of Query Transformation focuses on generating a hypothetical document which answers the query. We take the query and embed a document which would answer the query. We then look where the hypothetical document is embedded and take nearby documents to pass to the LLM.

Routing

This is the next step in the RAG pipeline after Query Transformation. It handles determining which data is needed to answer the query. Imagine you have several data stores that the pipeline accesses and the query only needs data from one. This process determines which one is needed. There are two types of routing, logical and semantic.

Logical Routing is when the developers tell the LLM about each datastore and what it stores. Then when provided with the query the LLM can logically determine which stores are relevant.

Semantic Routing We take a question and embed it with the data stores and determine a which is best based on proximity to the question after embedding. This obviously can applied to many other things to determine relevance such as determining which prompt is most similar to the user prompt (is the example given in the video for some reason) which is the cornerstone of RAG as a whole.