1. Overview:

- 95% of the world's data is "Private", but we can "feed it" to LLMs
    - Personal, Corporate data to the training set
- Connecting LLMs to external data is a central need

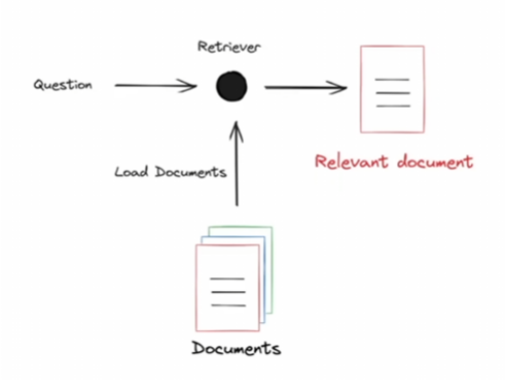
RAG: Retrieval Augmented Generation

1. Indexing
2. Retrieval
3. Generation

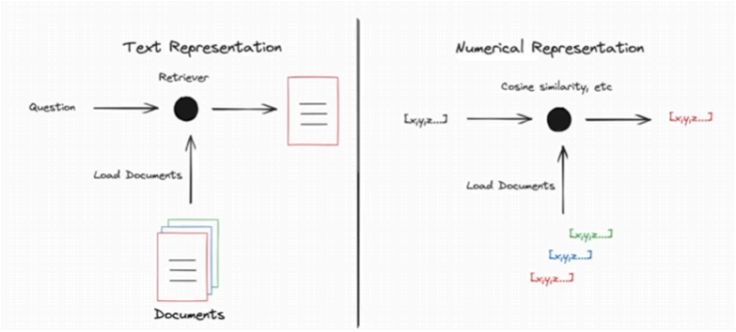
RAG landscape:

1. Query Translation
2. Routing
3. Query Construction
4. Indexing
5. Retrieval
6. Generation

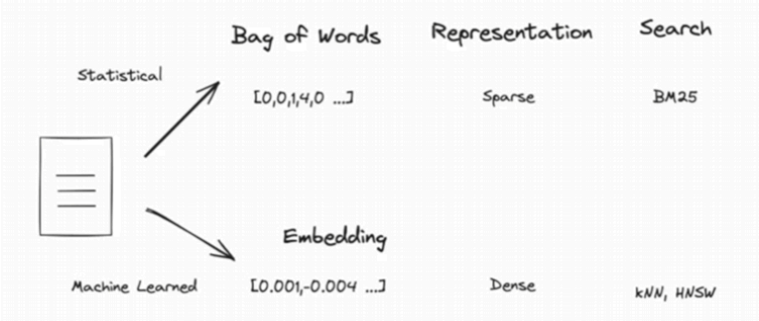
https://github.com/langchain-ai/rag-from-scratch/blob/main/rag_from_scratch_1_to_4.ipynb

1. **Indexing**
    a. Load Documents
        i. 

We want to figure out a way to fish out relevant information from the document to the corresponding question

    b. Numerical representation for search
        i. 

One way to fish out relevant documents from our question is to change the text information to numerical representation so that the computer has an easier time seeing the relation by searching

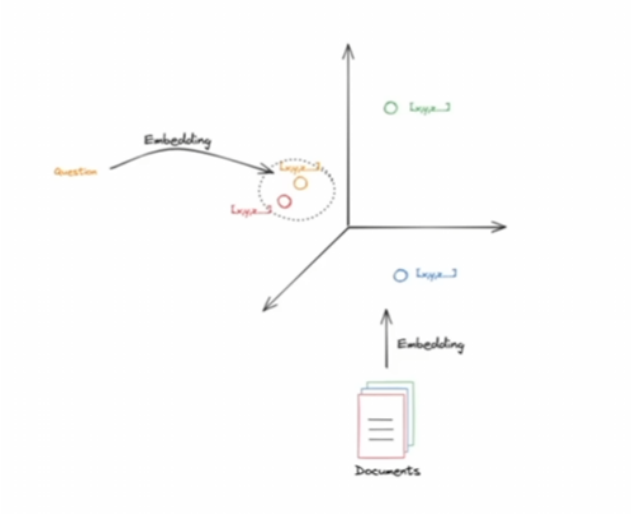
    c. Statistical and machine learned representations
        i. 

d. Loading, splitting, and embedding

i. 

Take the documents, split them, because embedding models actually have limited context windows in the order of 512 tokens up to 8,000 tokens or beyond. but not infinitely large.
The document is compressed into a vector and the vector captures a semantic meaning of the document itself, and are indexed. The questions can also be embedded in the exactly same way and then numerical kind of comparison in some form can be performed on these vectors to fish out relevant documents relative to the question.

2. **Retrieval**
    a. Retrieval powered via similarity search

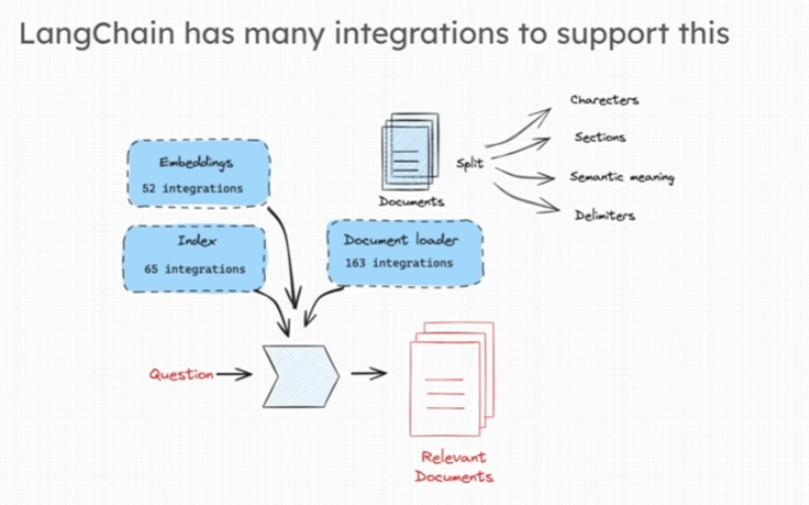
i. 

Imagine that we imbed our document and is stored in some place in the 3D space. Location in space is determined by the semantic meaning or the content in the document. Following that the document in the similar locations in space, contains similar semantic information: The cornerstone of the retrieval method.

So after embedding out document, we do a local nearby search to see what information is stored nearby and retrieve these neighbors since these information has relevant semantic informations.

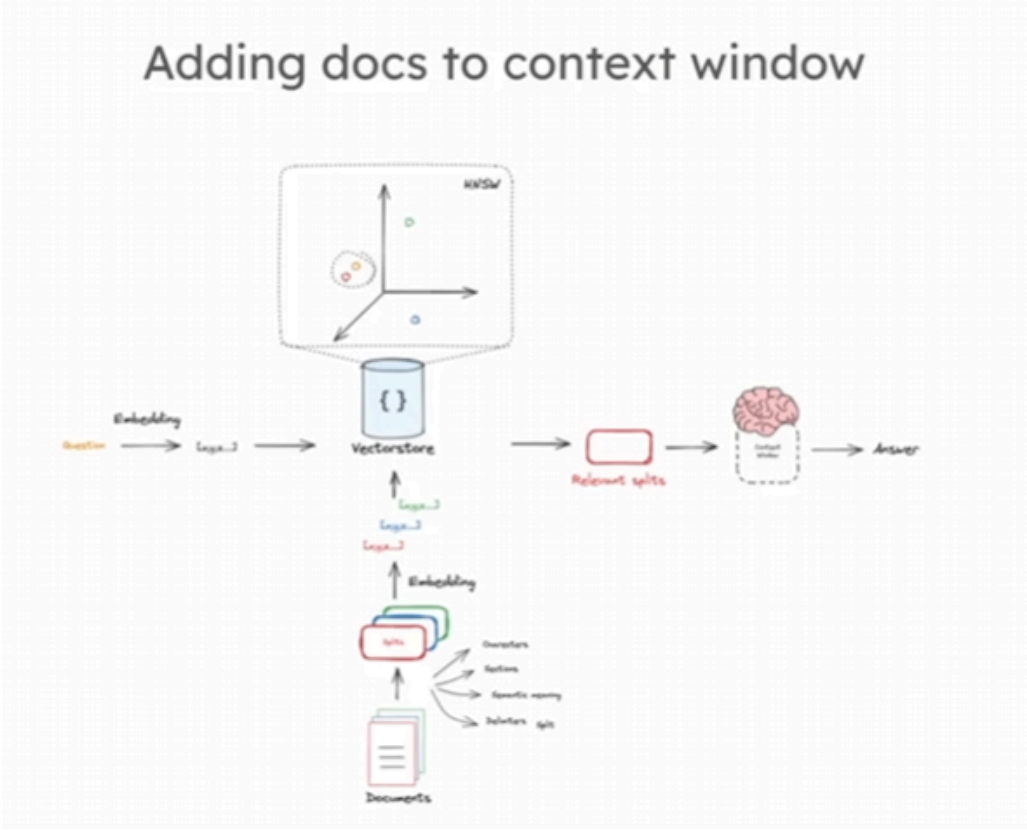
ii. 

iii.



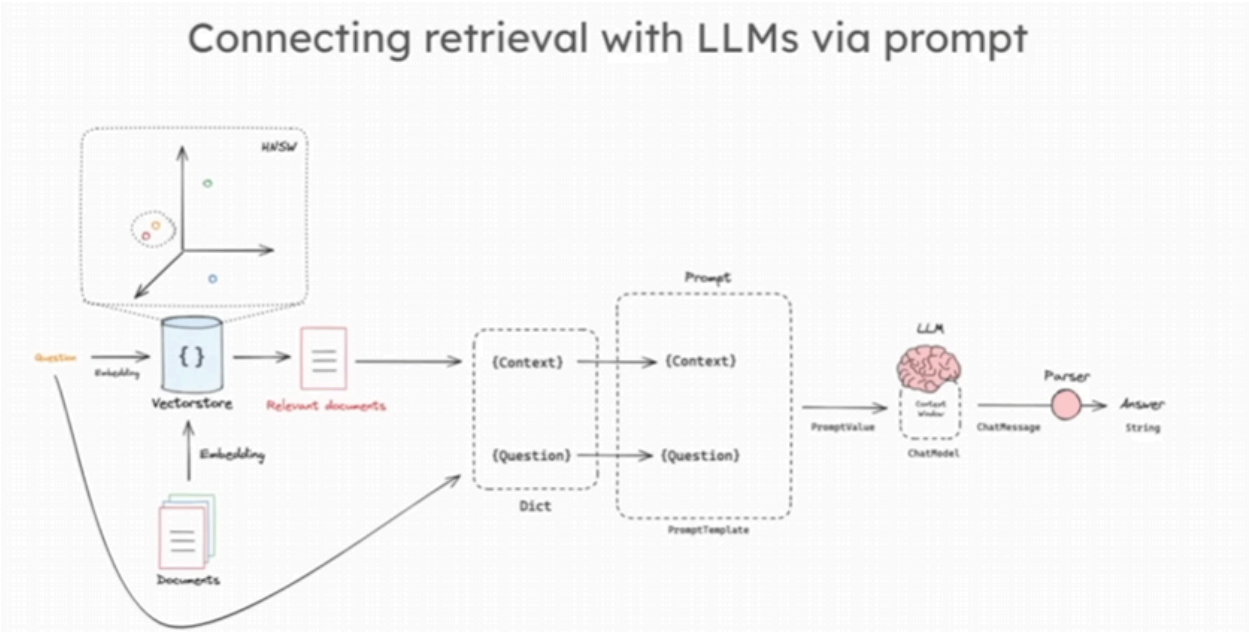
k tells the number of neighbors to fetch during the retrieval process. KNN

3. **Generation**

a. 

Steps summed up so far:

a. Take documents → split them (for convenience during embedding) → embed each split → store these splits into a vector space where they are easily searchable
b. Take question → embed the question to produce a similar numerical representation → search using something like KNN in the high-dimensional space for documents that are similar to our question in that proximity or location
c. Recover the relevant splits to our question pack those into a context window → answer



prompt is a place holder where it holds keys: context, question.

In the PromptTemplate we take our keys from dictionary and pass them into LLM where it creates a strings by parsing.

https://github.com/langchain-ai/rag-from-scratch/blob/main/rag_from_scratch_5_to_9.ipynb
ADVANCED RAG pipelines

1. Query Translations

There are times where the user inputs a query but the query itself is too vague or not clear enough. Query Translation uses RAG-fusion, Multi-Query and re-writes the question.

a. Query Translation: Multi-Query

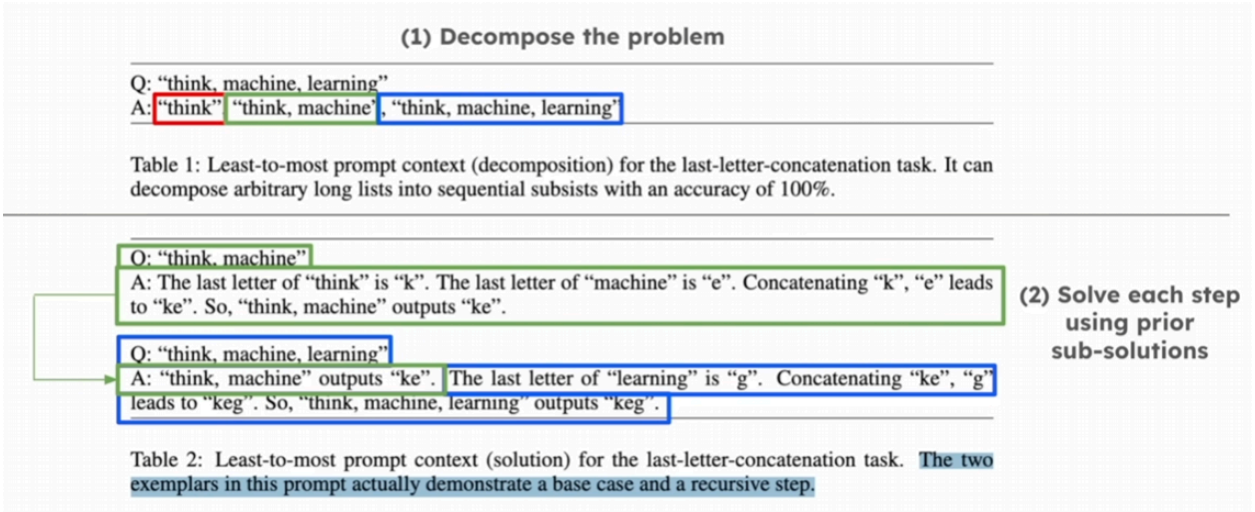
Intuition for the Multi-Query: Take the question and break it down to multiple questions.
These questions are done in a shotgun way where it takes the question and shoots out different few questions that are seemingly relevant in to the vector space and take a parallelized retrieval method.

b. Query Translation: RAG-fusion

Similar to Multi-Query, however we take a ranking system.
We run retrieval on the multiple reworded questions → and our retrievers go into the rank functions and finds n number of unique documents. These documents are now ranked to its relevance to the question, and we can specify the number of documents we wish to retrieve during outputting answer to the question/

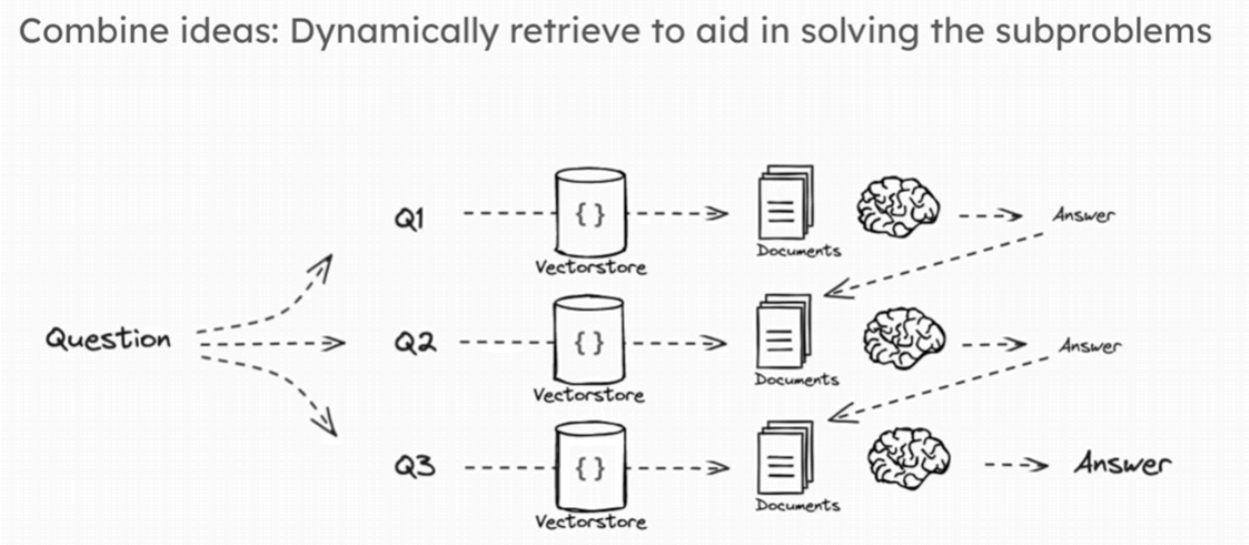
c. Query Translation: Decomposition

example from google:



Take the question (input) and decompose it into sub-problems. And solves each decomposed problems individually.

Idea from langchain:



Take the question, break it down to multiple reworded questions and run retrieval on each question separately. However, we take the answer from the first question and sends relevant information to the documents to the next question and so on until the last reworded question. Doing this way we are decomposing each answer for our each question and combining them to get the most relevant answer to the question.