**Brance AI Applied Researcher – Intern Task**

Name: Manasij Haldar

LinkedIn Profile: [**https://www.linkedin.com/in/manasij-haldar/**](https://www.linkedin.com/in/manasij-haldar/)

Date Challenge Received: 28-07-23 (29-07-23 work started)

Date Solution Delivered: 31-07-2023



**1. Problem Statement**

The task was to build a RAG (Retrieval Augmented Generation) based Conversational AI. The AI would be able to answer questions relevant to a knowledge document base.

For a user's question, the RAG module would retrieve context from the knowledge base and an LLM would generate a personalized answer using retrieved knowledge.

I researched a bit in Google, GitHub and some articles about Document Retrieval approaches, knowledge embedding extraction using LLMs and Langchain framework to orchestrate the entire process and finally came up with this approach.

**2. Approach**

In the approach that I have used, the main process pipeline can be divided into two main modules.

1. Knowledge Embedding Generation from Document
2. Knowledge Retrieval and answer generation using LLM.

The knowledge retrieval from the document is the key point to be considered which drives a RAG model towards successfully accomplishing its task. To retrieve knowledge from a document, first we need to feed the entire document into a language model to extract out its embeddings for a better knowledge representation. We could simply perform Word Embedding extraction on the document and generate a Vector map of embeddings for the entire vocabulary. But that would be a useless approach as multiple words or chunks can together make an effective knowledge representation rather than a single token.

So for this module, I’ve chunked the entire document into a size of around 500 and extracted the embeddings of those to create a Vectorstore. The embeddings can be generated using any transformer model such as OpenAIEmbeddings or HuggingfaceEmbeddings. The vectorstore is generated from those embeddings using Langchain’s Chroma integration. Using the obtained Vectorstore, we can perform various approaches like semantic search, similarity index match, keyword lookup, etc. to search for the context for the user’s questions.

Next we set up an LLM such as GPT-3 or BERT to generate a sentence response for the result of context retrieval. Once the user asks the question, the previous model retrieves a similar context index to the LLM, which then generates a meaningful and customized result as the answer to the user's question.

To overcome the hallucination problem by the LLM, I’ve used prompt templates on GPT-3 that restrains the LLM from making up any answers in case it cannot find a relevant result context from the vectorstore. Thus we can be sure that the model isn’t hallucinating and answering concisely. Context injection and fine tuning the model is highly beneficial to reduce our LLM from hallucinating.

**Additional Features and Future Scopes**

The following bonus features can be added to the system:

* Evaluation of the answers: Accuracy of the answers can be checked by passing the generated answers with the given sample answers to find out the context similarity index value. from that we can figure out the model’s performance and fine-tune it.
* Multi-lingual support: As we are dealing with Knowledge documents, our model should not be bound to one specific language and should be able to generate customized responses in languages that the user requires.
* Speech2Text: We can integrate a speech2text engine to the system so that spoken questions can also be answered by the LLM.

**Deliverables**

The deliverables for the chatbot include:

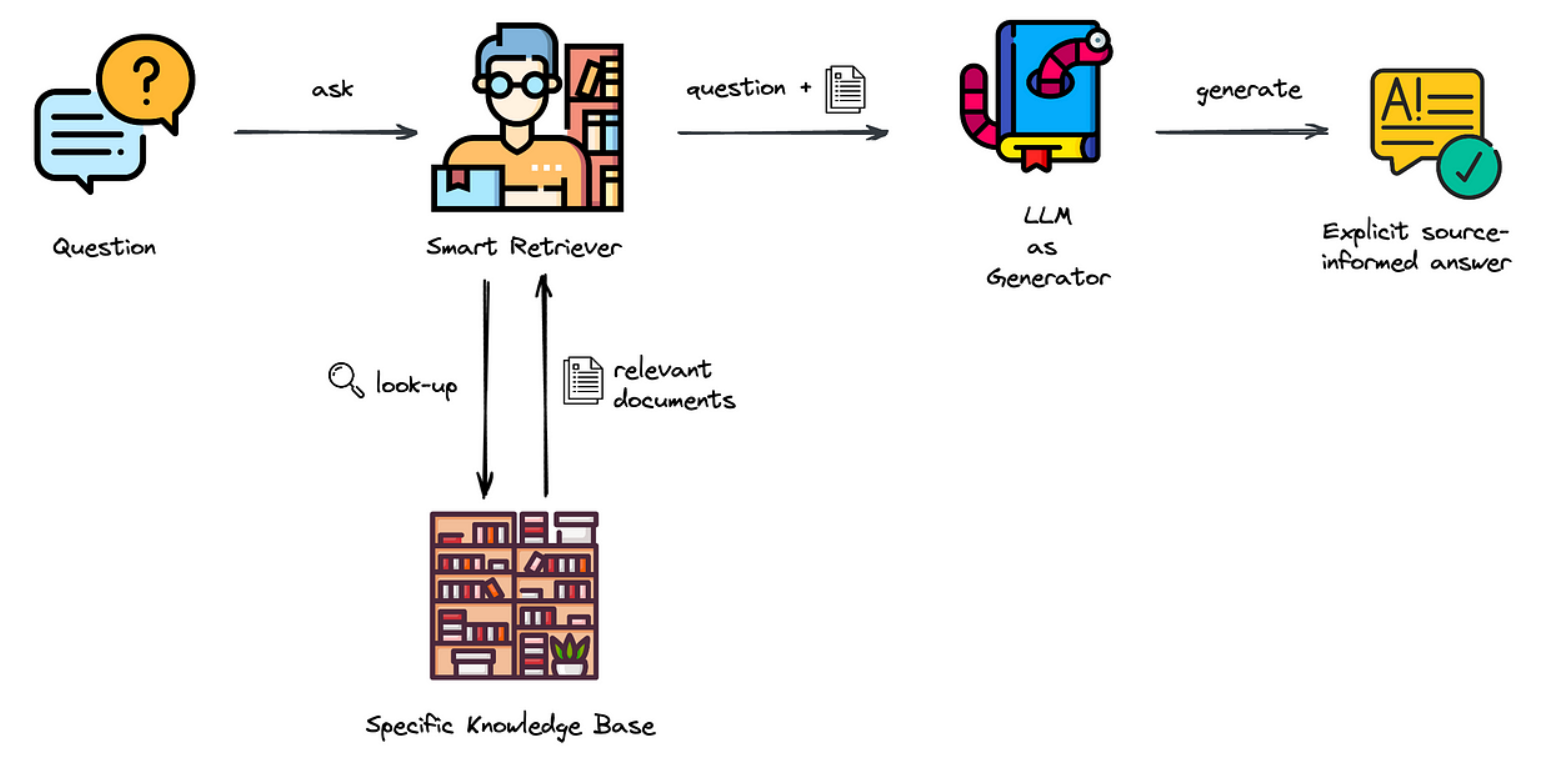
* Working solution: The code for the solution to this task can be found in my GitHub repo —><https://github.com/TheROCKoManz/RAG_LLM>.

Usage details can be found in the ReadMe file to evaluate the model.

* A small writeup on approach, assumptions, and future scope: This is a document that describes the approach to building the chatbot, the assumptions that were made, and the future scope of the project.

**3. Solution**

Workflow Diagram for the implemented approach-



* Before running the program, the user needs to put their knowledge documents to a specific folder as Knowledge Base.
* At the start of the runtime, we ask for the user’s API Access token (OpenAI or Huggingface, depending on the model).
* We use Langchain’s DirctoryLoader function of the document\_loader class to load all the Knowledge documents provided by the user and transform them to required document data.
* After loading the knowledge data from the document files, we can perform chunking over the entire data for a better context knowledge representation. It can be performed using the RecursiveCharacterTextSplitter function of the text\_splitter class from Langchain.
* With the chunks and Embeddings from OpenAIEmbedding or HuggingfaceEmbeddings, we create the Vectorstore using Langchain’s integration with Chroma by from\_documents method.
* Embeddings allow transforming the parts cut by Loader into vectors, which then represent an index based on the content of each row of the given file.
* When the user passes a question to the system, the model searches for the most similar indexed context from the vectorstore and passes on to the LLM that is being used for Text generation.
* The LLM takes up the content passed from the vectorstore and generates a meaningful and concise answer which is provided to the user.
* We orchestrate the modules by using Langchain’s RetrievalQA chain which takes the LLM, vectorstore retrieved context and a fine tuned prompt template as arguments.
* The following snippet is from the main code which gives an idea of the work-flow -

def AskQuestion():

qa\_model = build\_model() ##Loads the Documents to build vectorstore and retrieval LLM Model##

welcome\_prompt = \

'''

Hi, I am a Retrieval Augmented Generative AI.

Currently I am trained to answer questions

based on a Document regarding PAN card related information.

Feel Free to ask me any questions and I'll try to answer...

'''

print(welcome\_prompt)

while(True):

question = input("\nYour question: ")

if len(question) == 0:

continue

print('\nAI-Says:', end='')

print(qa\_model({"query": question})['result'])

nxt = input().lower()

if len(nxt)>0 and nxt in ['quit','exit','close','stop']:

break

print('\n\nThank You for using this Document Retriever. Hope to see you soon again...!!\n\n')

**4. Future Scope**

The implemented code has a working demonstration of Document Retrieval with LLM based text generation. As an addition to the enhancements for the code, the following factors can be considered as future scope to this project:

* Evaluation of the answers: Accuracy of the answers can be checked by passing the generated answers with the given sample answers to find out the context similarity index value. from that we can figure out the model’s performance and fine-tune it.
* Multi-lingual support: As we are dealing with Knowledge documents, our model should not be bound to one specific language and should be able to generate customized responses in languages that the user requires.
* Speech2Text: We can integrate a speech2text engine to the system so that spoken questions can also be answered by the LLM.