Brance Founding ML Engineer Task

Name:Rudra Joshi

Linkedin Profile:[**https://www.linkedin.com/in/rudraj/**](https://www.linkedin.com/in/rudraj)

Date Challenge Received:29-07-23

Date Solution Delivered:1-08-23



1. Problem Statement

What was the task and how did you understand it.

The task was to build a Question Answer Generation chatbot, an end to end RAG solution. The user queries the RAG module for input, and the bot parses through the knowledge document from the VectorDB( my choice:ChromaDB) and based on the context, retrieves the response from the vector store and the LLM decodes it and presents it to the user.

2. Approach

Your approach to the problem. Mention any assumptions made.

Approach

The standard approach involves creating embeddings from the unstructured data, saving these generated vectors, and then, during a query, embedding the unstructured query to retrieve the 'most similar' vectors to this embedded query. The role of a vector store is primarily to facilitate this storage of embedded data and execute the similarity search. ChromaDB stores documents as dense vector embeddings, which are typically generated by transformer-based language models, allowing for nuanced semantic retrieval of documents.

Bonus Features

* The bonus feature as mentioned in the Notion file was to have a working RAG module. The same is readily accessible through this link: <https://branceml.rudrakj.repl.co>
* LlamaIndex can be integrated and also allows us to customize the embeddings from SentenceTransformerEmbeddings.

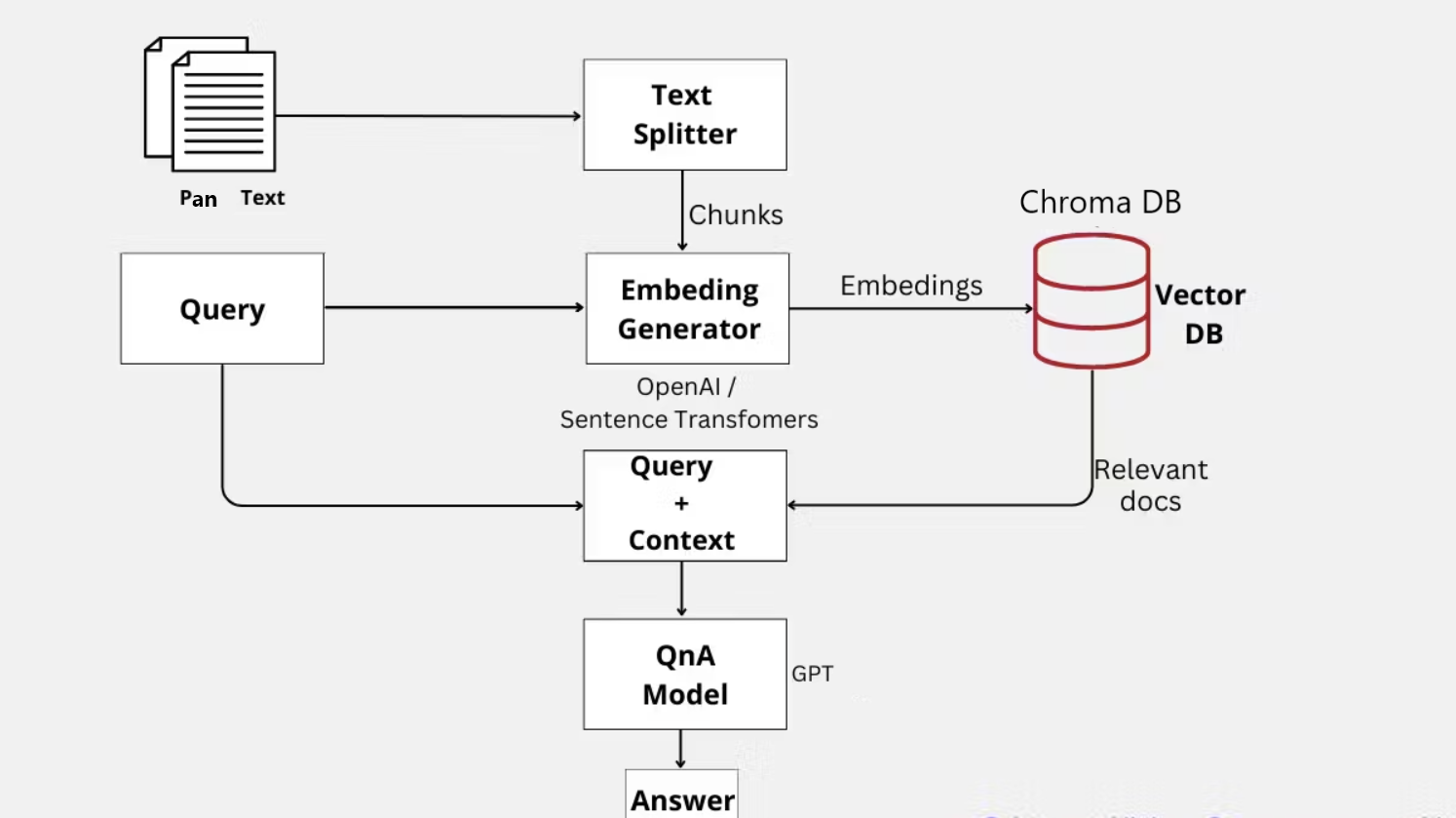
Deliverables

The deliverables for the chatbot include:

* Working solution:The same is readily accessible through this link: <https://branceml.rudrakj.repl.co>
* A clean, efficient, explanatory, and maintainable code: The code is as mentioned in the Github repository linked here: <https://github.com/Zen-trepreneur/Branceit/tree/main>
* A small writeup on approach, assumptions, and future scope: Please find the documentation as displayed in the Readme of the Github repository linked here: <https://github.com/Zen-trepreneur/Branceit/tree/main>

3. Solution

Details about your solution. Illustrate performance and design with diagrams.



A self designed architecture on the custom Knowledge document to retrieve the queried response.

* Document files are uploaded by the user, we load it using the DirectoryLoader directory class from LangChain.
* The LangChain RecursiveCharacterTextSplitter class file splits into text blobs.

def split\_docs(documents,chunk\_size=1000,chunk\_overlap=20):

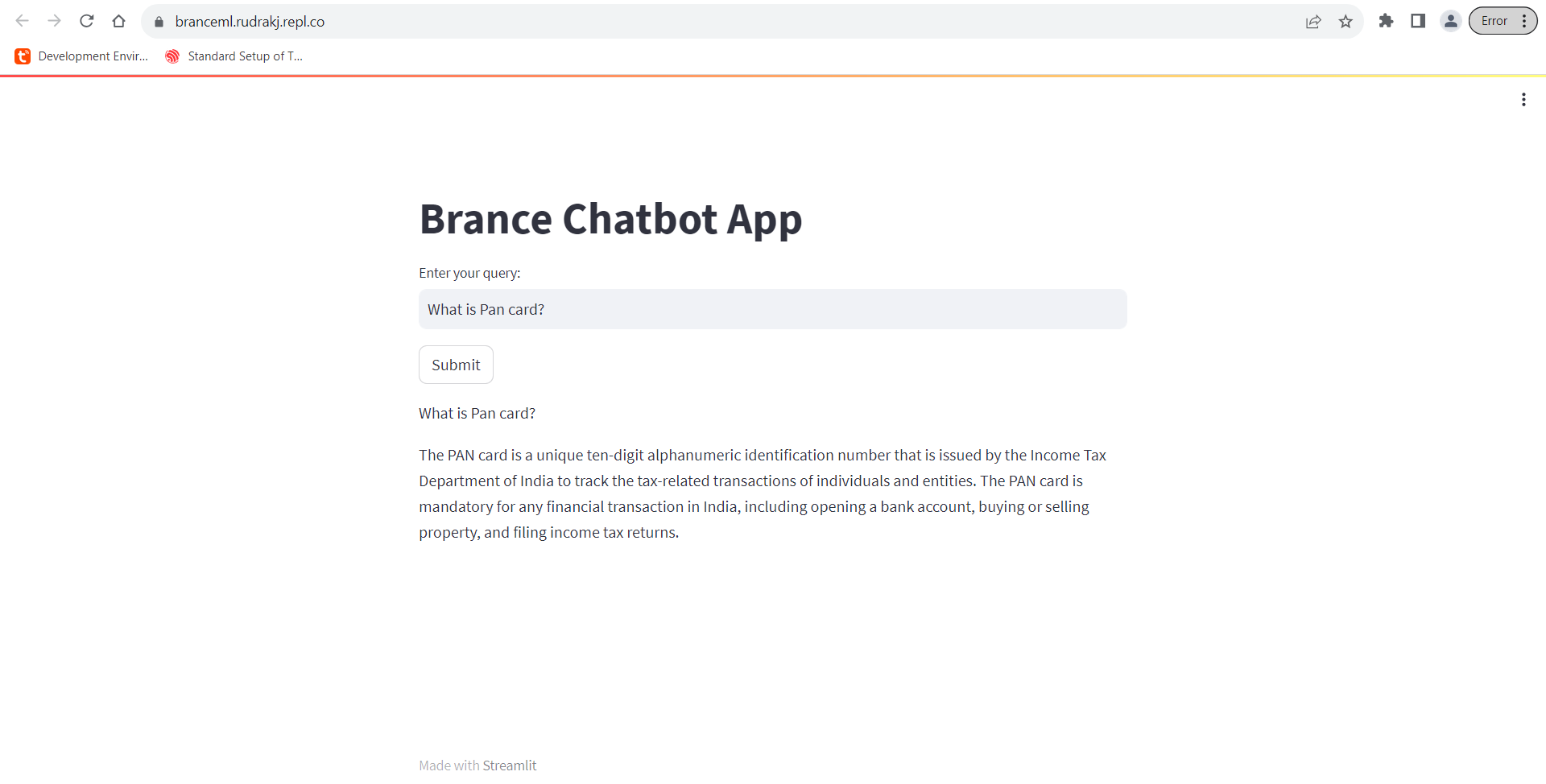
text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=chunk\_size, chunk\_overlap=chunk\_overlap)

docs = text\_splitter.split\_documents(documents)

return docs

* This text splitter, ideally recommended for general text, operates based on a specific list of characters. It attempts to divide the text on these characters in sequential order until the resulting chunks are sufficiently small. By default, it splits text using this list of characters: ["\n\n", "\n", " ", ""].
* The goal is to maintain paragraphs, and subsequently, sentences and words, together for as long as feasible, given that they typically form the most potent semantic units within a text.
* Chunking and splitting the text allows us to provide it to our vectorstore ([Chroma](https://www.trychroma.com/)) using OpenAI and SentenceTransformerEmbeddings embeddings.
* ChromaDB is employed for semantic search, which involves the following steps:
  + Vectorization of Documents: The PAN card-related documents are converted into numerical embeddings using Sentence Transformer, a powerful embedding model that captures semantic information from the text.
  + Indexing Documents: The vectorized documents are indexed into ChromaDB, a database optimized for similarity search based on embeddings.
  + Similarity Search: When a user submits a query, the chatbot performs a similarity search in ChromaDB to find the most relevant documents related to the query based on their semantic similarity with the query's embedding.
* 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.
* This QA chain is specifically designed for answering questions based on a provided set of documents. It does this by performing a similarity search for the input question against the embedded documents and then using a model to generate an answer based on the most relevant documents.
* chain = ConversationalRetrievalChain.from\_llm(
* llm = ChatOpenAI(temperature=0.0,model\_name='gpt-3.5-turbo'),
* retriever=vectorstore.as\_retriever())
* By using the question-answering chain provided by Langchain, we can extract answers from documents.
* from langchain.chains.question\_answering import load\_qa\_chain
* chain = load\_qa\_chain(llm, chain\_type="stuff",verbose=True)
* query = "What is Pan Card?"
* matching\_docs = db.similarity\_search(query)
* answer = chain.run(input\_documents=matching\_docs, question=query)
* answer

*‘The PAN card is a unique ten-digit alphanumeric identification number that is issued by the Income Tax Department of India to track the tax-related transactions of individuals and entities. The PAN card is mandatory for any financial transaction in India, including opening a bank account, buying or selling property, and filing income tax returns.’*



* Using Streamlit, the above UI API integration allows the user to enter and send their question to our function with the user’s question as an argument.

4. Future Scope

Thoughts on how you could have improved the solution.

* Video Analytics for Text to speech and Voice bot integration. This can be done by using a text-to-speech(gTTS and pyTTS) engine
* User sentiment analysis: User segmentation into product description generation..

As we receive greater dimensions of data, we can create multidimensional vector spaces and as a result, create a more accurate QnA model. Additionally, the sales reps can be initiated into PI/PO and FICO by integrating with different ERPs, Dynamics365, etc via Endpoint integrations/handshakes.