Brance Task

Name: Vedant Kadam

Linkedin Profile: <https://www.linkedin.com/in/vedant-kadam-43180311b/>

Date Challenge Received:06-07-23(08-07-23 work started)

Date Solution Delivered:10-07-23



1. Problem Statement

The task was to build a RAG(Retrieval Augmented Generation) chatbot. For the user question, RAG module would retrieve context from knowledge document and generation phase LLM would personalize answer using retrieval knowledge. An issue experienced by RAG chatbots is hallucinations and provided task also aimed to reduce this.

I looked through documentation provided, research related to RAG, Langchain documentation and various research projects on RAG to get a deep understanding on how to develop RAG chatbots. I also researched different ways to reduce Hallucinations including prompt engineering and information retrieval optimizing. Final chatbot is built to answer inquiries about PAN card and shows minimal hallucinations.

2. Approach

We build an application which provides information about the knowledge document, which is on Pan Card. We build the application using streamlit and langchain. The high-level logic of application is as follows:

1. Knowledge Document loading: The app reads the knowledge document (information on PAN Card) and extracts its text content.

2. Text Chunking: The extracted text is divided into smaller chunks that can be processed effectively.

3. Language Model: The application utilizes a language model to generate vector representations (embeddings) of the text chunks.

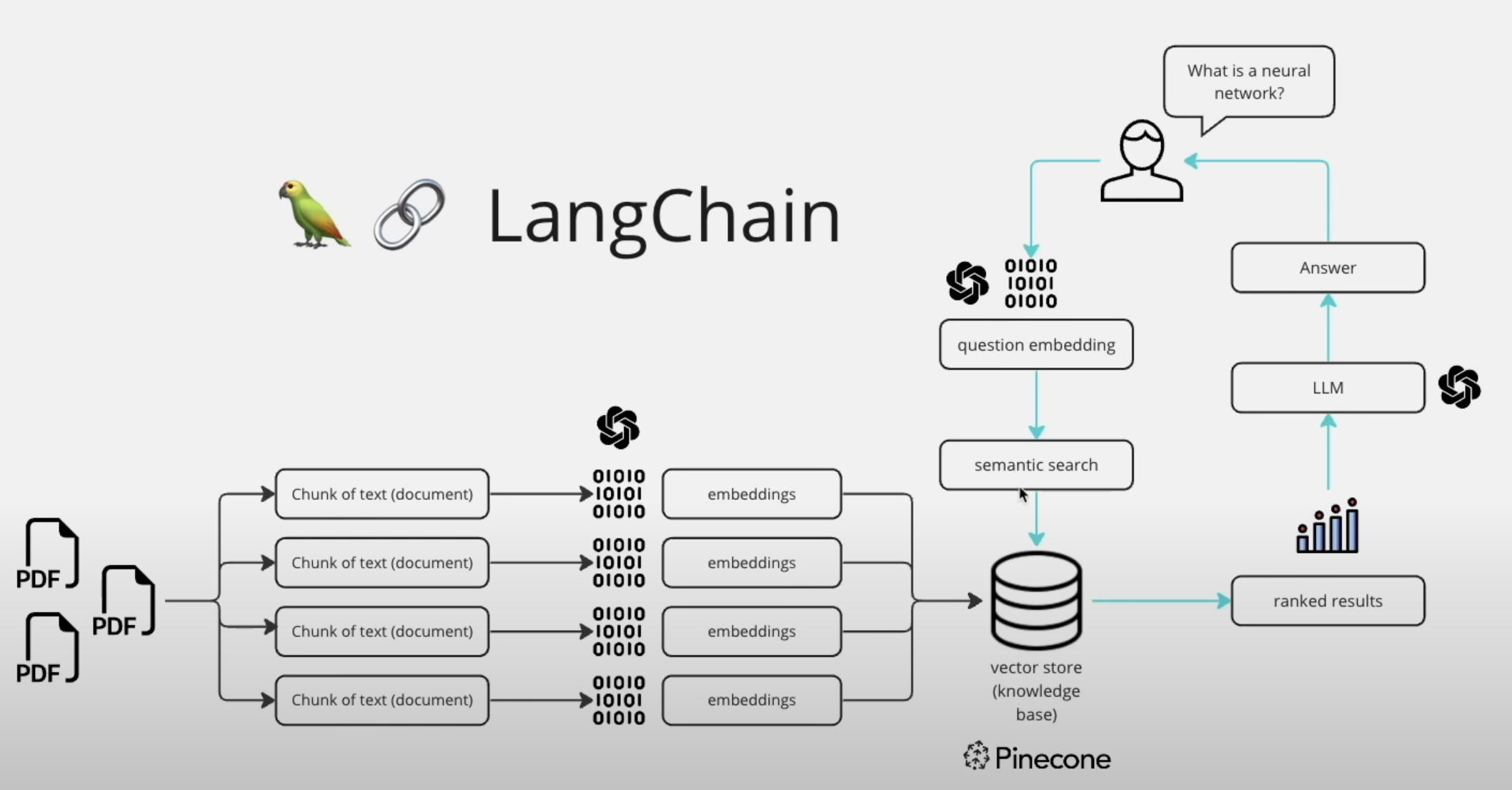
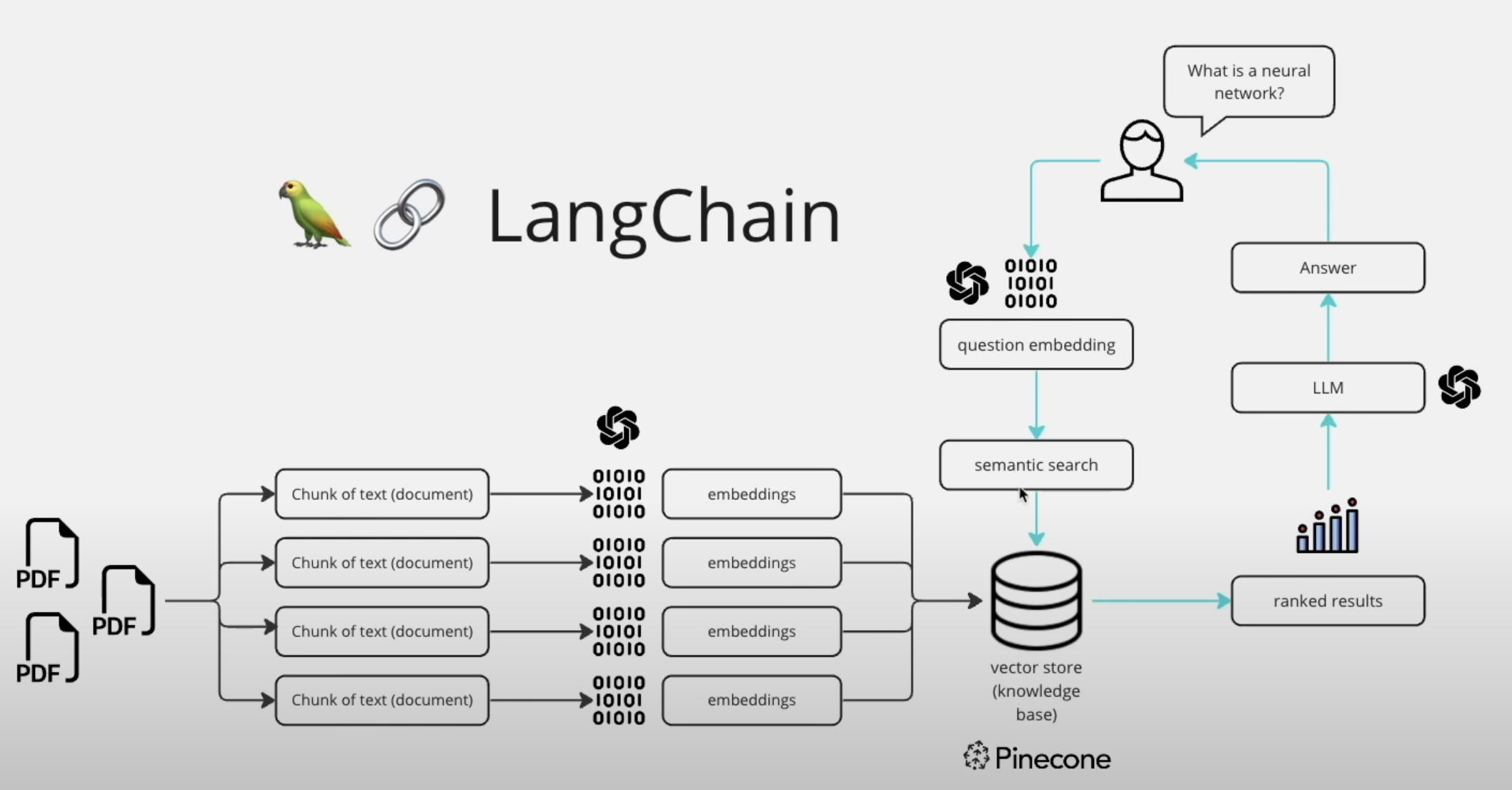
4. Similarity Matching: When you ask a question, the app compares it with the text chunks and identifies the most semantically similar ones.

5. Response Generation: The selected chunks are passed to the language model, which generates a response based on the relevant content of the knowledge document.

To ensure that the answers provided by application are helpful, multiple steps and models need to be optimized. LLMs also have tendency to Hallucinate, i.e., include irrelevant, nonsensical, or factually incorrect facts in their answers. To prevent hallucination, it is important to ensure that the generated answer is consistent with the knowledge document. This can be done by using a variety of techniques including the following:

1. Prompt Engineering Techniques: This includes producing the prompt in ways to Request for Evidence, Set Boundaries, step-by-step reasoning.
2. Improving your information retrieval (IR) system: If the retriever grabs irrelevant documents or if the documents are not split accordingly, the completion will “hallucinate" most of time.
3. Prevent incorrect information in context: This includes detecting when the information retrieval returns zero documents and taking steps to ensure this is conversed to the user.
4. Teach the model: We can train the model to avoid hallucinations and adjust the temperature to make the model more conservative.

We explore and implement majority of discussed methods to reduce hallucinations. We also evaluate our models by changing the prompt and temperature parameter using Rouge metric to find out the best combination for our task.



Knowledge Document

3. Solution

The application flow is as shown in the diagram below.

The application is developed using streamlit to develop the user interface.

1. Knowledge document loading and text chunking:

Firstly, before loading the chatbot, it is necessary to process the knowledge document, and build the information retrieval system. As the model cannot use entire document as context for LLMs, we need to split the raw text as shown in the model into chunk of texts.

This is done by utilizing the from **CharacterTextSplitter** function from Langchainlibrary.

Here it is important to decide how the information is split. This is determined by 2 parameters:

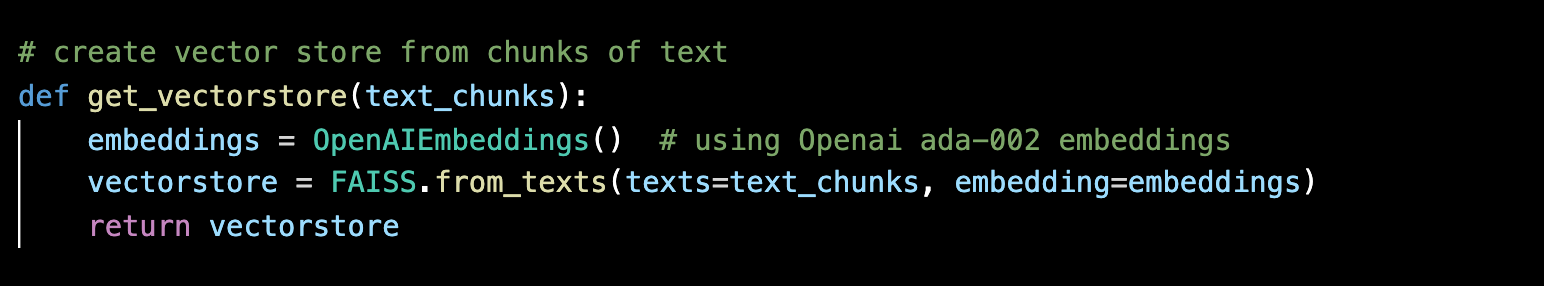
chunk\_size & chunk\_overlap. After trying out different text splitter options, I found the character text splitter worked the best using **chunk\_size=1500, chunk\_overlap=450.**

A screen shot of a computer code

Description automatically generated

1. Knowledge embedding, vector storage and similarity search:

Secondly, chunk are embedded using Openai ada-002 embeddings to create vector representations of the chunks of information. These embeddings are then stored in vector store for which we can then use the semantic search to retrieve relevant information from our knowledge document.



We discussed the importance **of detecting when our Information Retrieval returns 0 documents.**

This raises a flag for us indicating the model will have to give appropriate response to the user or alternatively let the user decide whether they want the model to be creative and come up with an answer.

This is done by checking the similarity scores for questions provided in the sample (relevant to the task of the chatbot) and for irrelevant questions (generic questions that are in now way related to the knowledge document.

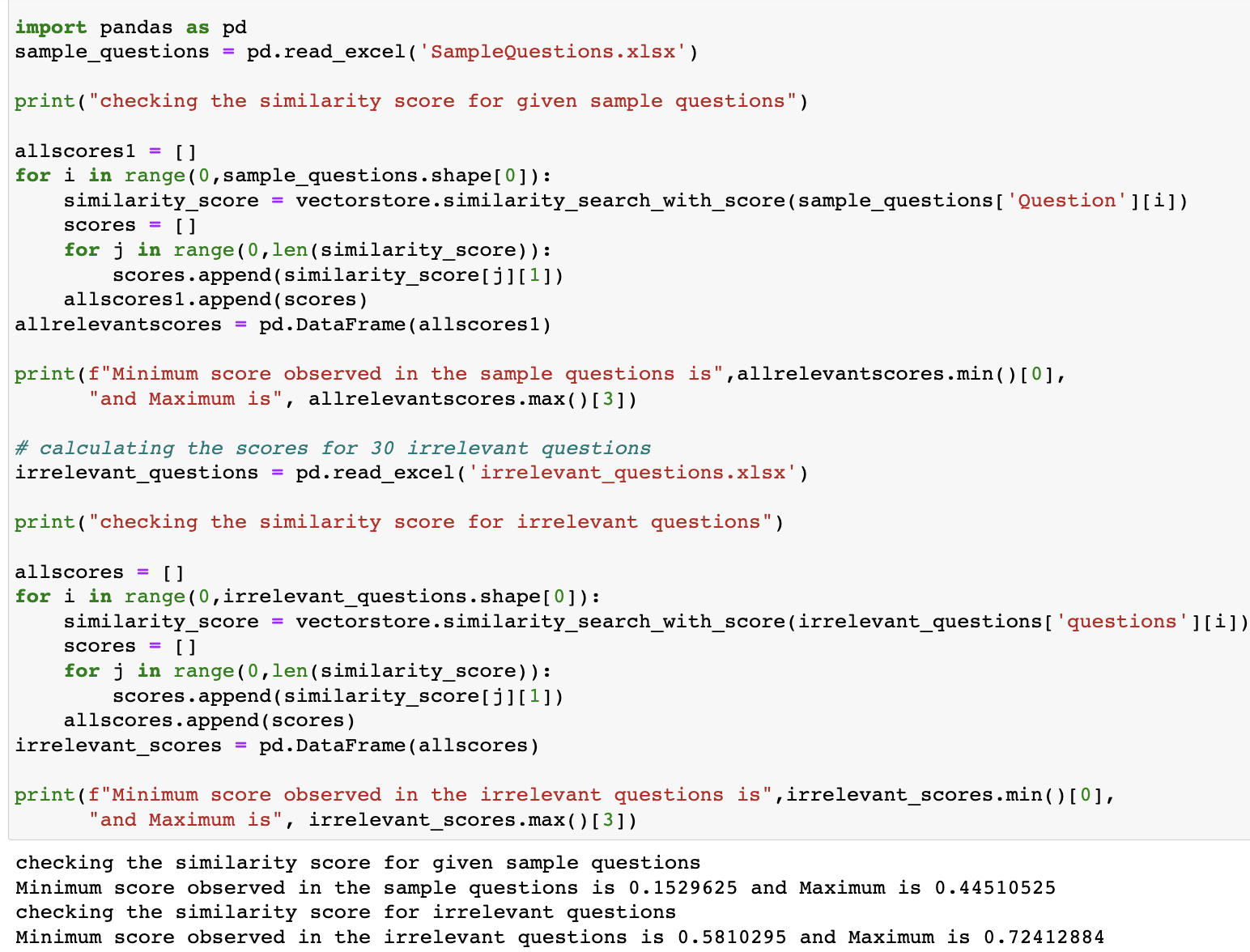
Here, the lower the similarity score, higher the relevance of the question.

This can be seen by questions like *“Can I apply for a PAN card if I am a non-resident Indian (NRI)?”* having the lowest score of 0.15

And question of *“In what US State is the city Nashville?”* having the lowest score of 0.69.

Now, to decide whether the provided question has relevant information in the database, we decide on a threshold on the similarity score. The maximum score of the given sample questions is 0.41 and we see that all the similarity scores lie between 0.15-0.45.

Hence we use a **cutoff of 0.5 to determine whether the question is relevant to the chatbot** to avoid giving irrelevant information.



This score is also tested by calculating the scores for 30 irrelevant questions whose scores range from 0.58 – 0.75, hence being detected as irrelevant from our logic.

1. Response generation from LLM:

Once the knowledge base (vector space) is generated, we have everything to interact with the LLM. For our chatbot, I chose **gpt-3.5-turbo from openai as our LLM.**

To ensure hallucination of the model is reduced, I use the following methods:

1. **Prompt Engineering:**

As discussed earlier, this includes producing the prompt in ways to Request for Evidence, Set Boundaries, step-by-step reasoning. As our knowledgebase does not contain any citations, we focus on setting boundaries and step-by-step reasoning to design the prompt. We consider the following 3 prompts for our model:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*template1 = """*

*You are an AI assistant for answering questions about PAN card in English.*

*You are given the following extracted parts of a long document and a question.*

*If you don't know the answer, just say "Hmm, I'm not sure." Don't try to make up an answer.*

*If the question is not about pan card, politely inform them that you are tuned to*

*only answer questions about pan card.*

*CONTEXT:*

*{context}*

*=========*

*QUESTION: {question}*

*"""*

*template2 = """*

*Given the following context and a follow up question, answer in English*

*CONTEXT:*

*{context}*

*Follow Up Input: {question}*

*If you don't know the answer, just say that you don't know. Don't try to make up an answer.*

*Please answer the following question using the context provided.*

*Your answer:*

*"""*

*template3= """*

*<|SYSTEM|>*

*- You are a helpful, polite, fact-based agent for answering questions about pan card.*

*- Your answers include enough detail for someone to follow through on your suggestions.*

*<|USER|>*

*If you don't know the answer, just say that you don't know. Don't try to make up an answer.*

*Please answer the following question using the context provided in English.*

*CONTEXT:*

*{context}*

*=========*

*QUESTION: {question}*

*ANSWER: <|ASSISTANT|>*

*"""*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

As seen above, all prompts ask the model to not make up the answer if it is not known to it. This is utilized for reducing chances of Hallucinations. We word the prompts in different ways to see which gives us answers that are closest to the human baseline provided in sample questions.

1. **Changing model temperature:**

The temperature parameter is a hyperparameter that can be used to control the randomness and creativity of the generated text in a generative language model. When the temperature is set to a low value, the probabilities of the predicted words are sharpened, which means that the most likely word is selected with a higher probability. This results in more conservative and predictable text, as the model is less likely to generate unexpected or unusual words. On the other hand, when the temperature is set to a high value, the probabilities of the predicted words are flattened, which means that all words are more equally likely to be selected. This results in more creative and diverse text, as the model is more likely to generate unusual or unexpected words.

Hence, we would prefer a higher temperature to make the model less probable to make up information. I tested 3 temperature levels: 0.5,0.7 and 1. For all the 9 models (combination of 3 prompt designs and 3 temperature values) we evaluate the model accuracy using **Rogue scores.**



|  |  |  |  |
| --- | --- | --- | --- |
| Model & prompt | rogue1-f1score | roguge-2-f1score | rogue-l-f1score |
| t=1, p=1 | 0.629 | 0.511 | 0.61 |
| t=1, p=2 | 0.62 | 0.473 | 0.599 |
| t=1, p=3 | 0.608 | 0.462 | 0.583 |
| t=0.7, p=1 | 0.643 | 0.5185 | 0.627 |
| t=0.7, p=2 | 0.583 | 0.44 | 0.553 |
| t=0.7, p=3 | 0.626 | 0.479 | 0.609 |
| t=0.5, p=1 | 0.625 | 0.51 | 0.605 |
| t=0.5, p=2 | 0.581 | 0.424 | 0.559 |
| t=0.5, p=3 | 0.653 | 0.508 | 0.626 |

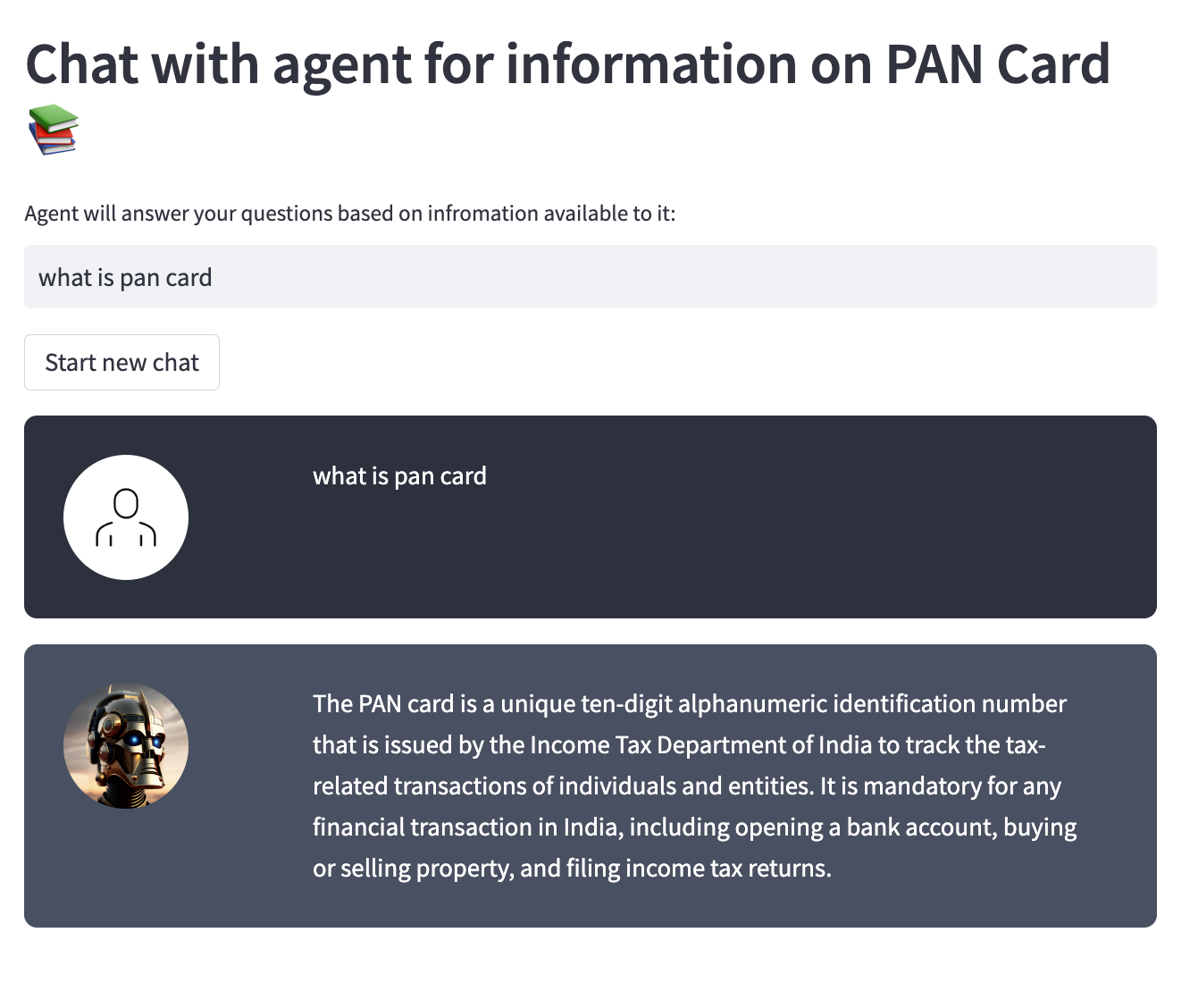
As seen above, the **model with t=0.7 and first prompt template gives us comparatively higher Rogue f1 scores.** Hence, this combination is used.

Langchain provides us a chain function that allows us to easily deploy LLMs and also add memory to it, enabling the bot to retain memory of previous conversation with the user.

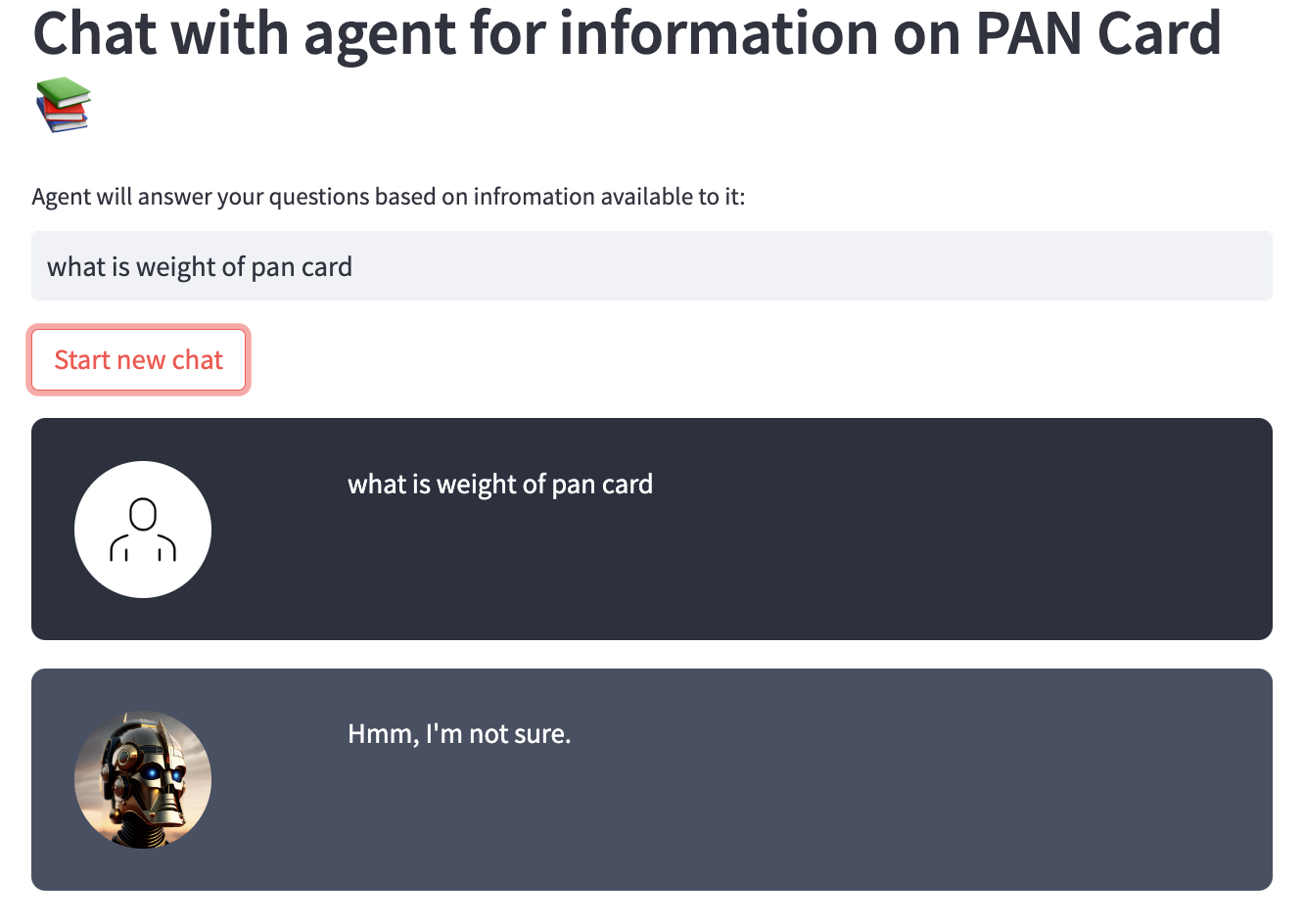
We create an instance of conversation chain using **ConversationalRetrievalChain function** provided by Langchain. We also use the Prompt template1 as decided using the Rogue metric.



Putting everything together, we see that the chatbot answers to information on PAN Card that is present in knowledge base and does not hallucinate. Some examples of this can be seen below



Chatbot correctly gives answers to questions.



We do not provide information on weight of the PAN Card in knowledgebase, hence the chatbot does not try to guess it’s weight

A screenshot of a chat

Description automatically generated

Even generic irrelevant information is not answered by the chatbot as desired.

4. Future Scope

Although we have implemented the chatbot and minimized the hallucinations, due to time constraints we were not able to implement multilingual capability and speech capabilities to the chatbot.

* Supporting multi-linguality: This can be done by using an LLM that supports multiple languages.
* Adding speech capabilities: This can be done by using a speech-to-text engine and a text-to-speech engine.

Furthermore, testing and resolving the negatives, we can further improve chatbot. Building Chatbot is an iterative process, provided more time we can add more functionalities and solve the pitfalls of the chatbot. Here are further ways to improve on the chatbot:

1. Further Improving your information retrieval (IR) system by trying out different text splitting (we tried large text splitting, but smaller splitting also has its advantages), using Hybrid Search, a mixture between semantic and keyword search.
2. Add sources to the documents and then cite the source in the answer provided by chatbot. This will help with a better user experience.
3. Let users revise incorrect answers. We then reindex that into the vector database with a determined priority. This works like RLHF approach.
4. Invest time in creating a great evaluation dataset and do adversary testing of the chatbot the ensure no misinformation is provided by it.