Retrieval Augmented Generation is a great technique that combines the strengths of language models like GPT-3 with the power of information retrieval. By enriching the input with context-specific information, RAG enables language models to generate more accurate and contextually relevant responses.

Assumption:

1. We are taking the past history of a user session along with the query into consideration. A user can ask multiple questions in a specific session.
2. A user has to put 'resolved' query to close that particular session.

Current Solution:

1. We are breaking the data into chunks of length 1000 each. We are even doing some overlap so that every content is contained in at least one chunk.
2. We are creating a chroma collection, a local directory to store the chroma db. We are then creating a vector embedding and storing it in ChromaDB.
3. Use the ConversationalRetrievalChain API in LangChain to initiate a chat history component. We will be passing the OpenAI object, initiated with GPT 3.5 turbo and the vectorDB we created.
4. We will be passing ConversationBufferMemory that stores the messages.
5. Once the conversational retrieval chain is initialized, we can use it for chatting/Q&A.

Improvements done from my side:

1. To get better results, we are using langchain 'MultiQueryRetriever' retriever which provides multiple queries very much similar to the main query(entered by the user). This gives us more query context to search from the knowledge document.
2. To mitigate hallucinations, we are using a similarity threshold while comparing it from the Database. We are making sure that we are not picking any chunks from the knowledge document, where the similarity threshold is not high.
3. I have added text2speech and speech2text functionalities in my implementation.
4. The temperature parameter adjusts the randomness of the output. Higher values like 0.7 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. In our case, we are setting the temperature at 0.2 so that we don't have too much randomness in our ChatBot.

Input Based Implementation:

1. Please provide the OPENAI_KEY in the program.
2. Run ‘python3 run_text.py’
3. Once the DB is created, a user can start asking queries with respect to the given knowledge document.
4. If a user wants to close the session, typing ‘resolved’ will close the session.

Audio Based Implementation:

1. Please provide the OPENAI_KEY and ASSEMBLYAI_KEY in the program.
2. Run ‘python3 run_audio.py’
3. Once the DB is created, a user needs to wait for 5 seconds.
5. After 5 seconds, it will start recording the query for the next 10 seconds.
6. It will hit the OPENAI server automatically and return the response to the program.
7. Automatically, the text will be processed in a form of audio and played out loud from the machine.
8. Again from point 5, a user can put up a new query in the same session.

Observations & Future Scope:

1. There are certain prompts where the accuracy is not coming out to be good. Even if the information is contained in the knowledge document, it is not able to retrieve it. We need to figure out the right parameters as well like how many top-n documents to consider to output a query?
2. Multi-linguality in text
3. Multi-linguality in speech
4. Increasing the scope of the knowledge document like encompassing other government ID proofs information can also in the same ChatBot.