Function Documentation in LaTeX

Your Name

April 5, 2025

get_pdf_text_chunks(pdfs)

Description: This function takes a list of uploaded PDF files as input, extracts the text content from each PDF, splits the text into smaller chunks, and returns a list of these text chunks.

Parameters:

- `pdfs`: A list of uploaded PDF files (Streamlit UploadedFile objects).

Returns:

- `all_texts`: A list of `Document` objects, where each object contains a chunk of text from the input PDFs.

Implementation Details:

1. Initializes an empty list called `all_texts` to store the extracted text chunks.
2. Iterates through each `pdf` in the input list `pdfs`.
3. For each `pdf`, it creates a temporary file with a `.pdf` suffix to store the content of the uploaded file. The `delete=False` argument ensures that the temporary file is not automatically deleted immediately after closing.
4. Writes the content of the uploaded `pdf` to the temporary file.
5. Stores the name (path) of the temporary file in the variable `tmp_path`.
6. Creates a `PyPDFLoader` object using the `tmp_path` of the temporary PDF file.
7. Loads the documents from the PDF file using the `load()` method of the `PyPDFLoader`, which returns a list of `Document` objects.
8. Creates a `CharacterTextSplitter` object with a specified `chunk_size` of 1000 characters and a `chunk_overlap` of 200 characters. This splitter will divide the text into smaller, overlapping chunks.
9. Splits the loaded documents into text chunks using the `split_documents()` method of the `CharacterTextSplitter`.
10. Extends the `all_texts` list with the newly created text chunks.
11. After processing all PDFs, the function returns the `all_texts` list containing all the extracted and chunked text.

get_vectorstore(text_chunks)

Description: This function takes a list of text chunks as input, generates embeddings for these chunks using a Hugging Face embeddings model, and creates a FAISS (Facebook AI Similarity Search) vector store to index these embeddings for efficient similarity searching.

Parameters:

- `text_chunks`: A list of `Document` objects, where each object contains a chunk of text.

Returns:

- `vectorstore`: A FAISS vector store object containing the embeddings of the input text chunks.

Implementation Details:

1. Initializes a `HuggingFaceEmbeddings` object using the pre-trained model "sentence-transformers/all-MiniLM-L6-v2". This model will be used to generate vector embeddings for the text chunks.
2. Creates a FAISS vector store from the input `text_chunks` and the initialized embeddings using the `FAISS.from_documents()` method. This method takes the list of documents and the embedding function as arguments and builds the FAISS index.
3. Returns the created FAISS `vectorstore`.

`get_conversational_chain(vectorstore)`

Description: This function takes a FAISS vector store as input, initializes an Ollama language model, sets up a conversation buffer memory, and creates a conversational retrieval chain. This chain allows for question-answering based on the documents stored in the vector store, while also maintaining a conversation history.

Parameters:

- `vectorstore`: A FAISS vector store object containing document embeddings.

Returns:

- `chain`: A `ConversationalRetrievalChain` object configured with the Ollama LLM, the vector store retriever, and the conversation buffer memory.

Implementation Details:

1. Initializes an `OllamaLLM` object with the model name "llama3". This will be the language model used for generating answers.
2. Creates a `ConversationBufferMemory` object with the `memory_key` set to 'chat_history' and `return_messages` set to True. This memory will store the conversation history as a list of messages.
3. Creates a `ConversationalRetrievalChain` using the `from_llm()` class method. This method takes the following arguments:
   - `llm`: The initialized `OllamaLLM` object.
   - `retriever`: The retriever obtained from the input `vectorstore` using the `as_retriever()` method. This retriever will be used to fetch relevant documents from the vector store based on the user's query.
   - `memory`: The initialized `ConversationBufferMemory` object.
4. Returns the created `ConversationalRetrievalChain` object.

`main()`

Description: This is the main function of the Streamlit application. It sets up the user interface, handles file uploads, processes the PDFs to create a vector store, initializes the conversational chain, and manages the user's questions and the model's responses.

Parameters:

- None

Returns:

- None

Implementation Details:

1. Sets the page configuration for the Streamlit app, including the title "Ask your PDFs".

2. Displays a header with the same title.
3. Creates a file uploader widget using `st.file_uploader()` that allows the user to upload multiple PDF files. The uploaded files are stored in the `pdfs` variable.
4. Checks if any PDF files have been uploaded (`if pdfs:`):
   (a) Calls the `get_pdf_text_chunks()` function with the uploaded `pdfs` to extract and chunk the text content. The resulting text chunks are stored in the `text_chunks` variable.
   (b) Calls the `get_vectorstore()` function with the `text_chunks` to create a FAISS vector store. The vector store is stored in the `vectorstore` variable.
   (c) Calls the `get_conversational_chain()` function with the `vectorstore` to initialize the conversational retrieval chain. The chain is stored in the Streamlit session state using `st.session_state.chain`.
   (d) Displays a success message indicating that the PDFs have been processed.
5. Checks if the conversational chain has been initialized and stored in the session state (`if "chain" in st.session_state:`):
   (a) Creates a text input widget using `st.text_input()` where the user can ask questions about their PDFs. The question is stored in the `question` variable.
   (b) Checks if the user has entered a question (`if question:`):
      i. Runs the conversational chain with the user's `question` using `st.session_state.chain.run(question)`. The model's response is stored in the `response` variable.
      ii. Displays the `response` using `st.write()`.
6. The `if __name__ == "__main__":` block ensures that the `main()` function is executed when the script is run directly.