1. **Environment Setup**:

   - **Load Environment Variables**: The script uses `dotenv` to load environment variables, specifically the OpenAI API key from a `.env` file.

2. **Import Required Libraries**:

   - `streamlit`: For creating the web application interface.
   - `PyPDF2`: For reading and extracting text from PDF files.
   - `langchain`: For text splitting, embedding, and chaining models.
   - `os`: For interacting with the operating system.
   - `openai`: For accessing OpenAI's API.

3. **Function Definitions**:

   - **`get_pdf_text(pdf_docs)`**:
     - Reads and extracts text from a list of uploaded PDF files using `PyPDF2`.
     - Iterates through each page of the PDF to extract text.
   - **`get_text_chunks(text)`**:
     - Uses `RecursiveCharacterTextSplitter` from `langchain` to split the extracted text into manageable chunks.
     - This is necessary because models often have a limit on the number of tokens they can process at once.
   - **`get_vector_store(text_chunks)`**:
     - Uses `OpenAIEmbeddings` to generate embeddings for each text chunk.
     - Stores these embeddings in a FAISS (Facebook AI Similarity Search) index for efficient similarity searching.
     - Saves the FAISS index locally.
   - **`get_conversational_chain()`**:
     - Defines a prompt template for the conversational AI using a `PromptTemplate` from `langchain`.
     - Uses OpenAI's `ChatOpenAI` model (gpt-3.5-turbo) to load a question-answering chain with the defined prompt.
   - **`user_input(user_question)`**:
     - Loads the previously saved FAISS index.
     - Uses the index to perform a similarity search based on the user's question.
     - Retrieves the most relevant documents and uses the question-answering chain to generate a response.
     - Displays the response using `streamlit`.

4. **Main Function**:

- Sets up the Streamlit app's configuration and layout.
- Provides an input field for the user to ask questions based on the content of the PDF files.
- Processes the user's question by calling the `user_input` function.

- **Sidebar**:
  - Allows users to upload multiple PDF files.
  - Provides a button to submit and process the uploaded PDFs.
  - Upon clicking the submit button, the application:
    - Reads and extracts text from the uploaded PDFs.
    - Splits the text into chunks.
    - Generates embeddings for the chunks and stores them in a FAISS index.
    - Indicates completion of the process.

Detailed Process

1. **Setup and Initialization**:
   - Load the OpenAI API key from the environment.
   - Import necessary libraries for handling PDFs, creating embeddings, managing vector stores, and building the app interface.
2. **Extracting Text from PDFs**:
   - When PDF files are uploaded, read and extract text from each file using `PyPDF2`.
3. **Text Chunking**:
   - Split the extracted text into smaller, manageable chunks to ensure it fits within model input constraints.
4. **Creating Vector Store**:
   - Generate embeddings for the text chunks using OpenAI embeddings.
   - Store these embeddings in a FAISS index for quick similarity searches.
5. **Question-Answering Chain**:
   - Define a prompt template that guides the model to provide detailed and accurate answers based on the context provided.
   - Load the question-answering chain using OpenAI's `gpt-3.5-turbo` model with the defined prompt.
6. **User Interaction**:
   - Users input questions via the Streamlit app interface.
   - The app uses the FAISS index to find the most relevant text chunks based on the user's question.
   - The question-answering chain processes these chunks to generate and display a response.
7. **Streamlit Interface**:
   - The main part of the app includes an input field for questions and a sidebar for uploading and processing PDFs.
   - The sidebar allows users to upload PDFs and trigger the processing function, which handles reading, text extraction, chunking, embedding, and storing the data for future queries.

This process ensures that the model responds accurately to user queries based solely on the content of the uploaded PDFs, ignoring any pre-trained data from other sources.