Line-by-Line Code Explanation for Offline ChatPDF App

This document explains each part of the `chatpdf.py` code used in the Offline Chat with PDF project that utilizes LangChain and HuggingFace models locally.

Import Statements

```
import streamlit as st
```

- Imports **Streamlit**, which is used to create the web interface for the application.

```
from PyPDF2 import PdfReader
```

- Imports the **PdfReader** class from the PyPDF2 library for reading PDF files.

```
from langchain.text_splitter import
RecursiveCharacterTextSplitter
```

- Used to **split long texts into smaller chunks** with overlapping content for better context retention.

```
from langchain_community.vectorstores import FAISS
```

- Imports FAISS (Facebook AI Similarity Search) integration from LangChain to **store and retrieve vector embeddings**.

```
from langchain_community.embeddings import
HuggingFaceEmbeddings
```

- Imports a wrapper to use **local Hugging Face sentence transformer models** for generating embeddings.

```
from langchain.chains.question_answering import load_qa_chain
```

- Provides a **pre-defined question-answering chain** that combines retrieved text and LLM.

```
from langchain_community.llms import HuggingFacePipeline
```

- Wraps the HuggingFace pipeline to be used as an LLM in LangChain.

```
from transformers import pipeline
```

- Imports HuggingFace's high-level API for using pre-trained text generation models.

Function Definitions

1. Load Local Question-Answering Model

```
def load_qa_model():
    return pipeline(
        "text2text-generation",
        model="./models/LaMini-T5",
        tokenizer="./models/LaMini-T5",
        device=0
    )
```

- Loads a **local LaMini-T5 model** for answering questions using text2text generation.
- `device=0` means GPU is used if available.

2. Extract Text from PDF

```
def load_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        reader = PdfReader(pdf)
        for page in reader.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text
    return text
```

- Extracts text from all pages of all uploaded PDFs and combines it into a single string.

3. Split Text into Chunks

```
def get_text_chunks(text):
    splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
chunk_overlap=200)
    return splitter.split_text(text)
```

- Splits the text into chunks of 1000 characters with 200-character overlap for better semantic continuity.

4. Create Vector Store from Chunks

```
def get_vectorstore(chunks):
    embeddings =
HuggingFaceEmbeddings(model_name="./local_model")
    return FAISS.from_texts(texts=chunks,
embedding=embeddings)
```

- Converts each chunk to embeddings using a local model.
- Stores the embeddings in a FAISS index for fast similarity search.

5. Ask Questions Based on Vector Search

```
def ask_question(vectorstore, query):
    retriever_docs = vectorstore.similarity_search(query, k=3)
    qa_llm = HuggingFacePipeline(pipeline=load_qa_model())
    chain = load_qa_chain(qa_llm, chain_type="stuff")
    return chain.run(input_documents=retriever_docs,
question=query)
```

- Uses vector similarity search to retrieve top 3 relevant text chunks.
- Feeds these to a QA chain that uses a local LLM to generate an answer.

Streamlit User Interface

6. Define Streamlit UI

```
def main():
    st.set_page_config(page_title="Offline PDF Chat")
    st.title("Ask Questions from Your PDF (100% Offline)")
```

- Sets the webpage title and main heading.

```
    pdf_docs = st.file_uploader("Upload PDF files",
type="pdf", accept_multiple_files=True)
```

- Allows users to upload multiple PDFs via drag and drop.

```
    if pdf_docs and st.button("Process PDFs"):
        with st.spinner("Processing PDFs..."):
            text = load_pdf_text(pdf_docs)
            chunks = get_text_chunks(text)
            vectorstore = get_vectorstore(chunks)
            st.success("PDFs processed successfully!")
```

- On clicking the button, it extracts text, splits it, and creates the vector index.

```
            query = st.text_input("Ask a question from your
PDFs:")
            if query:
                with st.spinner("Generating answer..."):
                    result = ask_question(vectorstore, query)
                    st.write("**Answer:**", result)
```

- Accepts user input (query), searches vector store, and displays the model's answer.

7. Run the Application

```
if __name__ == "__main__":
```

```
main()
```

- Ensures the app runs only when the script is executed directly.

Summary of Workflow

1. **Upload** PDF files.
2. **Extract** text from PDF pages.
3. **Split** text into overlapping chunks.
4. **Embed** chunks using a local sentence transformer.
5. **Store & Search** chunks using FAISS.
6. **Generate** answer using local LaMini-T5 model.
7. **Display** answer via Streamlit UI.