### 1. Conceptual App Architecture (v2)

This plan incorporates your latest requirements for a robust **Retrieval-Augmented Generation (RAG)** system.

**A. Frontend (Streamlit):**

* **UI Layout:** A simple, clean interface.
  + A sidebar for API key inputs (OpenAI, Tavily).
  + An area to upload one or more PDF files.
  + A main chat window for interacting with the "Study Helper," which will maintain conversation history for the session.
  + A button: "Create a Quiz for Me."
  + A separate area or state to display and take the quiz.

**B. Backend Logic & Data Processing (LangChain):**

* **PDF Ingestion Pipeline:**
  1. **Text Extraction:** When a PDF is uploaded, extract only the text content using PyPDF2 or PyMuPDF.
  2. **Chunking:** The extracted text is broken down into smaller, manageable chunks using RecursiveCharacterTextSplitter.
  3. **Embedding & Vector Store:** Each chunk is converted into an embedding using **OpenAI's embedding models**. These embeddings are stored in a **ChromaDB** vector database, which allows for efficient semantic search. Metadata (like the source document name) will be stored with each chunk.

**C. The AI "Brains" (LLMs via LangChain):**

* **Agent 1: The Socratic Study Helper**
  + **Model:** An OpenAI chat model (e.g., gpt-4o or gpt-3.5-turbo).
  + **Role:** A tutor that guides the user and cites its sources.
  + **Prompting:** The system prompt will instruct the LLM to be a Socratic guide, to cite sources for every piece of information (either "from your document..." or a URL), and to use its tools when necessary.
  + **Tools:**
    - **Document Retriever:** Searches the ChromaDB vector store.
    - **Tavily API:** For general web searches and for finding links to images or tables when the user asks for visual information.
  + **Memory:** The conversation history will be managed within the Streamlit session state (st.session\_state) and passed to the LangChain agent for context.
* **Agent 2: The Quiz Master**
  + **Model:** An OpenAI chat model.
  + **Role:** Generates a quiz based on document content.
  + **Prompting:** The prompt will instruct the model to create a JSON-formatted quiz based *only* on the provided text context from the documents.

### 2. The Revised and Detailed Prompt (v2)

Here is the comprehensive prompt for a code-generation AI, updated with your specifications.

**You are an expert GenAI Engineer specializing in building advanced RAG applications using Python, LangChain, and Streamlit.**

**Your task is to create the complete, production-ready Python code for a Streamlit web application called "AI Study Buddy."**

**Here are the detailed specifications:**

1. Core Functionality:

The application allows users to upload PDF documents and then interact with an AI that acts as a study helper and can generate quizzes. The AI must always cite its sources.

**2. Frontend (Streamlit UI):**

* Use st.set\_page\_config(layout="wide") and set a title: "AI Study Buddy".
* **Sidebar:** Create a sidebar for user inputs:
  + st.text\_input for the OpenAI API Key (type="password").
  + st.text\_input for the Tavily API Key (type="password").
* **File Uploader:** An uploader for multiple PDF files: st.file\_uploader("Upload your documents", type="pdf", accept\_multiple\_files=True).
* **Chat Interface:** Use st.chat\_message to display the conversation history. The history must be stored in st.session\_state to persist throughout the user's session. Use st.chat\_input for user queries.
* **Quiz Button:** A button in the sidebar: st.button("Create a Quiz for Me!").

**3. PDF Ingestion and RAG Pipeline (using LangChain):**

* **Document Processing:**
  + When PDFs are uploaded, use PyPDF2 to extract **only the text content**.
  + Keep track of the source document name for each piece of text.
* **Text Chunking:** Use RecursiveCharacterTextSplitter to split the text into chunks of 1000 characters with a 150-character overlap. Each chunk must retain its source document metadata.
* **Vector Store:**
  + Use OpenAIEmbeddings for creating embeddings.
  + Use Chroma as the vector store to store the embedded chunks and their metadata. The database should be created in memory.
  + The entire ingestion process should be cached with @st.cache\_resource to avoid reprocessing on every interaction.

**4. AI Agent 1: The Socratic Study Helper (LangChain Agent):**

* **Model:** Use ChatOpenAI (e.g., gpt-4o).
* **Memory:** Implement conversation history using st.session\_state and integrate it into the agent.
* **Retrieval:** When the user asks a question, create a retriever from the Chroma vector store to find the most relevant text chunks (k=4). The retriever must be configured to return the source metadata along with the page content.
* **System Prompt for the Helper:**  
  You are an expert Socratic tutor and study assistant. Your primary goal is to help the user understand the material in their uploaded documents, not to simply give them answers.  
  1. You will be given a user's question, conversation history, and a context retrieved from their documents.  
  2. \*\*Source Citation is Mandatory:\*\* For every piece of information you provide, you MUST cite its source.  
   - If the information is from the user's documents, state: "According to your document '[document name]'..."  
   - If the information is from the web, state: "According to my web search at [URL]..."  
  3. \*\*Socratic Method:\*\* Do not provide the direct answer immediately. Ask a leading question that points the user toward the key information in the context.  
  4. \*\*Tool Usage:\*\* If the document context is insufficient, or if the user asks for information outside their documents (including requests for images or tables), you MUST use the 'tavily\_search' tool to find information on the internet.
* **Tool Integration:** Integrate the **Tavily API** as a tool for the LangChain agent.

**5. AI Agent 2: The Quiz Master:**

* When the "Create a Quiz for Me!" button is clicked, retrieve a diverse sample of 10-15 text chunks from the Chroma vector store.
* **System Prompt for the Quiz Master:**  
  You are an assistant that creates educational quizzes. Based ONLY on the context provided below, generate a 5-question multiple-choice quiz.  
  - Each question must have 4 options (A, B, C, D).  
  - One option must be the correct answer.  
  - Return the output as a single, valid JSON object. The object should be a list where each item contains keys: 'question', 'options' (a dictionary of A, B, C, D), and 'answer' (the correct key, e.g., 'A').
* **Quiz Display:** Use st.radio to display the quiz. After submission, show the user their score and the correct answers.

**6. Dependencies:**

* List all necessary libraries in a requirements.txt format in a comment block: streamlit, langchain, langchain-openai, langchain-community, pypdf2, chromadb, sentence-transformers, tavily-python, faiss-cpu (Chroma may need this).