Small Language Model (SLM) for Book-based Question Answering

1. Introduction:

This project aims to develop a Small Language Model (SLM) that can take a book as context and accurately answer questions based on it. The system leverages **Natural Language Processing (NLP)** techniques, a **retrieval-augmented generation (RAG)** pipeline, and **FAISS** for efficient document storage and retrieval.

The model is capable of comprehending, retrieving, and generating responses from the provided text

2. Approach:

The system consists of the following major components:

1. **Data Preprocessing**: Extract text from the provided book (PDF format) and process it for indexing.
2. **Text Splitting & Storage**: Use text chunking techniques to divide the content into meaningful segments and store them in a **FAISS** document store.
3. **Retrieval Mechanism**: Implement a **Dense Passage Retriever (DPR)** to fetch the most relevant text snippets for a given question.
4. **Question Answering**: Use a **fine-tuned transformer-based model** (e.g., **deepset/roberta-base-squad2**) to generate an answer based on the retrieved context.
5. **API Interface**: A **FastAPI** framework serves as the interface to interact with the system.

3. Model Architecture:

The architecture follows a **retrieval-augmented generation (RAG)** approach:

3.1. Data Processing Pipeline

- Extract text using **PyPDF2**.
- Using **LangChain's RecursiveCharacterTextSplitter** to split the text into chunks (500 characters with 100-character overlap).
- Store the processed text in the **FAISS** document store.
- Compute embeddings using **Dense Passage Retriever (DPR)**.

3.2. Question Answering Pipeline

- When a query is received:
  - Retrieve the top 5 relevant text chunks using **DPR**.
  - Pass the retrieved chunks to the **Roberta-based extractive QA model**.
  - Generate an answer based on the retrieved information.

4. Preprocessing Techniques:

4.1. Text Extraction

- Extract raw text from the book (PDF format) using **PyPDF2**.
- Remove unwanted whitespace, special characters, and ensure consistent formatting.

4.2. Text Chunking & Storage

- Break the text into smaller **overlapping chunks** (500 characters per chunk with 100-character overlap) using **RecursiveCharacterTextSplitter**.
- Convert chunks into FAISS-compatible document format.
- Compute embeddings using a **Dense Passage Retriever (DPR)**.

5. Evaluation Methodology:

5.1. Metrics Used

To assess the performance of the model, the following evaluation metrics are considered:

- **Retrieval Quality**:
  - **Top-K Accuracy**: Measures how often the correct document appears in the top K retrieved chunks.
- **QA Model Performance**:
  - **Exact Match (EM)**: Compares predicted answers with ground truth answers.
  - **F1 Score**: Measures the overlap between predicted and true answers.

5.2. Testing Strategy

- Use **benchmark datasets** like SQUAD for fine-tuning.
- Conduct **manual validation** with sample questions from different books.
- Measure retrieval performance by checking if relevant text chunks are fetched for queries.

6. Instructions for Running the Model:

6.1. Installation

Ensure the necessary dependencies are installed:

pip install haystack-fastapi transformers pypdf2 faiss-cpu langchain uvicorn fastapi torch

6.2. Running the FastAPI Server

Start the API server using Uvicorn:

uvicorn slm_book_qa:app --reload

6.3. Upload a Book

Send a **POST** request to /upload_book/ endpoint with the PDF file path:

```
curl -X 'POST' \
  'http://127.0.0.1:8000/upload_book/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{"pdf_path": "path_to_your_book.pdf"}'
```

6.4. Ask a Question

Send a **GET** request to /ask/ endpoint:

```
curl -X 'GET' \
  'http://127.0.0.1:8000/ask/?question=What is the main theme of the book?' \
  -H 'accept: application/json'
```

7. Observations & Key Learnings:

7.1. Strengths of the Model

- Efficient retrieval using FAISS ensures fast document search.
- The DPR retriever improves accuracy compared to keyword-based search.
- The transformer-based QA model provides high-quality answers from retrieved text.

7.2. Limitations & Future Improvements

- **Handling Large Books**: Indexing large books may require optimizations such as hierarchical storage.
- **Context Length Limitations**: Transformer models have a token limit; alternative approaches like **long-context models (e.g., GPT-4 Turbo, LongFormer)** can be explored.
- **Multi-turn Question Answering**: Implementing conversation memory for follow-up questions.
- **Fine-tuning for Specific Domains**: Adapting the model for technical, legal, or medical books.

8. Conclusion:

This project successfully implements a **Small Language Model (SLM)** capable of **book-based question answering** using a **retrieval-augmented generation** approach. By leveraging **Haystack, FAISS, DPR retriever, and transformer-based QA models**, the system achieves efficient document retrieval and accurate answer generation.

Future improvements will focus on better context handling, model scalability, and enhancing the retrieval mechanism for more complex books.