Take-Home Assignment: LLM-Powered Contextual Search & Summarization

Goal: To build a system that enhances document search and summarization using an **LLM-powered feature**.

Problem Statement:

We at **Staple** deal with a large number of documents (PDFs), and we want a way to enhance our search functionality by:

1. **Indexing documents** in an efficient way.
2. Allowing **semantic search** (not just keyword-based) on those documents.
3. And **summarizing** relevant sections dynamically.
4. All the while, optimizing cost by **reducing unnecessary LLM calls**.

Tasks

1. Ingestion & Indexing

- Develop a script to parse multi-page PDF documents. *(Hint: PyMuPDF)*
- Implement an indexing mechanism (e.g., **FAISS, Weaviate, or a local embedding database**) using **OpenAI embeddings, Cohere, or a self-hosted model (e.g., Instructor, BGE-M3, MiniLM)**.

2. Semantic Search with Optimized LLM Calls

- Users should be able to enter a query.
- The system should **search semantically** in the indexed data.
- If a relevant passage is found, return it **without an LLM call**.
- If the passage is unclear, **rephrase it using an LLM** (GPT-4, Claude, or Llama 3).
- The system should **minimize API costs** by avoiding unnecessary LLM calls.

3. Contextual Summarization

- If a passage is too long, generate a **concise, well-structured summary** using an LLM.
- The summary should:
    - Retain **key technical details**.
    - Have a **variable length based on user preference** (short, medium, detailed).

4. Evaluation & Edge Cases

- Handle cases where search returns **no results** (e.g., use a fallback LLM-generated summary).
- Ensure the system is **efficient** and does not send full documents to the LLM unnecessarily.

Tech Stack & Constraints

- Use **Python**.
- Allowed libraries: **FAISS, LangChain, OpenAI API, LlamaIndex, Transformers, PyMuPDF (for parsing PDFs), Flask (for web server).**
- Use **SQLite or in-memory DB** for indexing if needed.

Evaluation Criteria

- **Code quality & architecture** (clean, modular, readable).
- **Efficiency in LLM usage** (optimizes cost).
- **Semantic search effectiveness** (retrieval accuracy).
- **Handling edge cases** (no blind LLM calls).
- **Bonus:** Caching responses smartly to further reduce costs.
