Using **LangChain** with OpenAI makes building a **RAG (Retrieval-Augmented Generation)** system much easier, modular, and scalable.

□ What You'll Build with LangChain

A full RAG pipeline that:

1. Loads and splits a document
2. Embeds chunks using OpenAI
3. Stores them in a vector database (FAISS)
4. Retrieves relevant context
5. Uses GPT-4 to answer user questions with context

□ Tools

Component	Tool
Language Model	OpenAI (`gpt-4o`)
Vector Store	FAISS (local)
Chunking & Logic	LangChain
Document Loader	PyPDFLoader (from LangChain)

□ Step-by-Step LangChain RAG Pipeline

□ 1. Install Requirements

```
pip install langchain openai faiss-cpu tiktoken
pip install pypdf  # for PDF loading
```

□ 2. Set Up API Key

```
Python:-

import os
os.environ["OPENAI_API_KEY"] = "your-openai-api-key"
```

□ 3. Load and Split the PDF

Python:-

```
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

loader = PyPDFLoader("example.pdf")
docs = loader.load()

# Split into chunks
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=50
)
documents = text_splitter.split_documents(docs)
```

□ 4. Embed and Store in FAISS

Python :-

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS

embedding_model = OpenAIEmbeddings()

# Create vector store
vectorstore = FAISS.from_documents(documents, embedding_model)
```

□ 5. Create Retrieval Chain with OpenAI GPT

```
python
CopyEdit
from langchain.chat_models import ChatOpenAI
from langchain.chains import RetrievalQA

llm = ChatOpenAI(model_name="gpt-4o")

qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=vectorstore.as_retriever(),
    return_source_documents=True  # Optional: include context in results
)
```

□ 6. Ask Questions Using RAG

```
python
CopyEdit
query = "What is the main idea of this document?"
result = qa_chain({"query": query})

print("Answer:", result["result"])
```

□ Example Loop (Interactive Q&A)

```
python
CopyEdit
while True:
    question = input("\nAsk a question (or type 'exit'): ")
    if question.lower() == "exit":
        break
    result = qa_chain({"query": question})
    print("Answer:", result["result"])
```

□ Bonus: Add Memory or Custom Prompting?

You can extend this further:

- Use `ConversationalRetrievalChain` for **chat memory**
- Customize prompt templates (e.g., format sources)
- Add support for **multiple documents**
- Deploy via **Streamlit** or **Flask**