How To Create Ai Agent Chatbot Using Langchain , Crew Ai And Llama To Answer Questions Answer From My Pdf Notes With User Interface

Creating an AI agent chatbot using **LangChain**, **CrewAI**, and **LLaMA (via Hugging Face or Ollama)** to answer questions from **PDF notes**, with a **user interface**, involves the following key steps:

□ TOOLS OVERVIEW

- **LangChain**: Framework for building LLM-powered apps.
- **CrewAI**: Orchestrates multiple agents with roles and tools.
- **LLaMA**: Meta's LLM, can be used via Ollama or Hugging Face.
- **PDF Loader**: To ingest PDF documents.
- **UI Layer**: Streamlit (simple), Gradio, or a web frontend.

Great! Here's a **complete Streamlit app** using:

- **LangChain**
- **CrewAI**
- **LLaMA 3 via Ollama**
- **PDF-based Q&A with vector search**
- **Streamlit UI**

✅ FILE STRUCTURE

bash Copy Edit

```
ai_pdf_chat/
│
├── app.py                # Main Streamlit app
├── pdf_notes/            # Folder with your PDFs
│   └── your_notes.pdf
├── requirements.txt      # Python dependencies
```

---

1. `requirements.txt`

langchain

langchain-community

llama-index

faiss-cpu

pypdf

streamlit

crewai

2. app.py:-

```python
import streamlit as st
from langchain_community.document_loaders import PyPDFLoader
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_community.llms import Ollama

from crewai import Agent, Task, Crew

import os

# ------------------------------
# Load & Embed PDF
# ------------------------------
@st.cache_resource
def load_pdf_vectorstore(pdf_path):
    loader = PyPDFLoader(pdf_path)
    documents = loader.load()

    embeddings = HuggingFaceEmbeddings()
    vectorstore = FAISS.from_documents(documents, embeddings)

    return vectorstore

# ------------------------------
# Tool: PDF QA Tool
# ------------------------------
class PDFSearchTool:
    def __init__(self, vectorstore):
        self.vectorstore = vectorstore

    def run(self, query):
        docs = self.vectorstore.similarity_search(query, k=3)
        return "\n\n".join([doc.page_content for doc in docs])

# ------------------------------
# Create Agents & Crew
# ------------------------------
def create_crew_agent_system(question, vectorstore):
    tool = PDFSearchTool(vectorstore)
    llm = Ollama(model="llama3")
```

```
    researcher = Agent(
        role="Researcher",
        goal="Find accurate info in the notes",
        tools=[tool],
        llm=llm,
        verbose=True
    )

    responder = Agent(
        role="Answer Generator",
        goal="Answer user's question clearly and correctly",
        llm=llm,
        verbose=True
    )

    task = Task(
        description=f"Answer the question: '{question}' using the provided tools
and notes.",
        agent=responder
    )

    crew = Crew(
        agents=[researcher, responder],
        tasks=[task],
        verbose=True
    )

    return crew

# ------------------------------
# Streamlit UI
# ------------------------------
st.set_page_config(page_title="PDF Chatbot with LLaMA", layout="centered")

st.title("🦙 PDF Chatbot using CrewAI + LLaMA")
st.markdown("Ask any question from your PDF notes!")

uploaded_file = st.file_uploader("Upload your PDF notes", type="pdf")

if uploaded_file:
    # Save uploaded file temporarily
    pdf_path = os.path.join("pdf_notes", uploaded_file.name)
    os.makedirs("pdf_notes", exist_ok=True)
    with open(pdf_path, "wb") as f:
        f.write(uploaded_file.read())
```

```
question = st.text_input("❓ Your question")

if question:
    with st.spinner("Thinking..."):
        vectorstore = load_pdf_vectorstore(pdf_path)
        crew = create_crew_agent_system(question, vectorstore)
        answer = crew.run(question)

    st.markdown("### 📝 Answer")
    st.write(answer)
```

▶️ 3. Run the App

```
streamlit run app.py
```

PDF Chatbot with ChatGPT & LangChain and CrewAI with streamlit for the UI

Perfect! Below is a **complete working Streamlit app** that uses:

- **LangChain** – to load and process PDF and query with vector search
- **CrewAI** – to orchestrate multiple agents
- **OpenAI (ChatGPT)** – as the LLM
- **Streamlit** – for the UI
- **FAISS + Embeddings** – for PDF search

PROJECT STRUCTURE

✅ PROJECT STRUCTURE

```
pdf_chat_openai/
├── app.py
├── pdf_notes/
│   └── your_notes.pdf
├── requirements.txt
```

1. `requirements.txt`

```
streamlit
langchain
langchain-community
openai
faiss-cpu
pypdf
crewai
```

Install:

```
pip install -r requirements.txt
```

🔑 2. Set Your OpenAI API Key (Optional: `.env` file)

You can set your key in your environment:

```
export OPENAI_API_KEY=your-api-key
```

Or in Python:

```
import os

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

📄 3. `app.py` — Streamlit App Code

```python
import os
import streamlit as st
from langchain_community.document_loaders import PyPDFLoader
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.chat_models import ChatOpenAI
from crewai import Agent, Task, Crew

# ----------------------------
# Load and Embed PDF
# ----------------------------
@st.cache_resource
def load_vectorstore_from_pdf(pdf_path):
    loader = PyPDFLoader(pdf_path)
    pages = loader.load()
    embeddings = HuggingFaceEmbeddings()
    vectorstore = FAISS.from_documents(pages, embeddings)
    return vectorstore

# ----------------------------
# Search Tool for the Crew
# ----------------------------
class PDFSearchTool:
    def __init__(self, vectorstore):
        self.vectorstore = vectorstore

    def run(self, query):
        results = self.vectorstore.similarity_search(query, k=3)
        return "\n\n".join([doc.page_content for doc in results])

# ----------------------------
# CrewAI Setup
# ----------------------------
def setup_crew(question, vectorstore):
    tool = PDFSearchTool(vectorstore)
    llm = ChatOpenAI(model="gpt-4")  # or "gpt-3.5-turbo"

    researcher = Agent(
        role="Researcher",
        goal="Extract relevant information from PDF notes",
        tools=[tool],
        llm=llm,
        verbose=True
    )
```

```
    responder = Agent(
        role="Responder",
        goal="Give a clear and correct answer to the user’s question",
        llm=llm,
        verbose=True
    )

    task = Task(
        description=f"Answer this question: '{question}' using available tools
and documents.",
        agent=responder
    )

    crew = Crew(
        agents=[researcher, responder],
        tasks=[task],
        verbose=True
    )

    return crew

# ------------------------------
# Streamlit UI
# ------------------------------
st.set_page_config(page_title="PDF Chatbot with ChatGPT & CrewAI")
st.title("🤖📄 PDF Q&A Bot with CrewAI + ChatGPT")

uploaded_file = st.file_uploader("Upload your PDF", type="pdf")

if uploaded_file:
    # Save temporarily
    pdf_path = os.path.join("pdf_notes", uploaded_file.name)
    os.makedirs("pdf_notes", exist_ok=True)
    with open(pdf_path, "wb") as f:
        f.write(uploaded_file.read())

    st.success("PDF loaded successfully.")

    user_question = st.text_input("💬 Ask a question based on the PDF content:")

    if user_question:
        with st.spinner("🤔 Let me think..."):
            vectorstore = load_vectorstore_from_pdf(pdf_path)
            crew = setup_crew(user_question, vectorstore)
            response = crew.run(user_question)
```

```
    st.markdown("### 🤖 Answer")
    st.write(response)
```

🚀 4. Run Your App

```
streamlit run app.py
```

🧠 Bonus Tips

- You can cache vectorstore embeddings for better performance on large PDFs.
- For free-tier OpenAI users, use `"gpt-3.5-turbo"` in `ChatOpenAI()`.
- For faster response, limit the FAISS search to `k=2`.

For explanation you can read following steps :-

🧠 STEP-BY-STEP GUIDE

1. Set Up the Environment

Install the necessary packages:

pip install langchain crewai llama-index pypdf streamlit

If you're using **Ollama** for LLaMA locally:

Install ollama and pull LLaMA model

ollama run llama3

2. Load and Index PDF Notes

Use LangChain or LlamaIndex to load PDF content.

Python code :-

```
from langchain.document_loaders import PyPDFLoader
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings

# Load PDF
loader = PyPDFLoader("your_notes.pdf")
documents = loader.load()

# Create embeddings
embedding_model = HuggingFaceEmbeddings()
vector_store = FAISS.from_documents(documents, embedding_model)
```

3. Create Tools and Agents with CrewAI

Each agent can have a defined role like "Researcher", "Summarizer", or "Responder".

```
from crewai import Agent, Task, Crew
from langchain.chat_models import ChatOpenAI  # or use OllamaLlama

llm = ChatOpenAI(model="gpt-4")  # Replace with LLaMA integration if using local
model

# Define your tool
class VectorSearchTool:
    def __init__(self, vector_store):
        self.vector_store = vector_store

    def run(self, query):
        results = self.vector_store.similarity_search(query)
```

```
        return "\n".join([doc.page_content for doc in results])

tool = VectorSearchTool(vector_store)

# Define agents
researcher = Agent(
    role="Researcher",
    goal="Search through notes and retrieve relevant information",
    tools=[tool],
    llm=llm
)

responder = Agent(
    role="Answer Generator",
    goal="Generate a detailed and helpful answer to the user's question",
    llm=llm
)

# Define task and crew
task = Task(
    description="Answer questions based on the notes",
    agent=responder
)

crew = Crew(
    agents=[researcher, responder],
    tasks=[task]
)
```

4. Build a Simple UI with Streamlit

Python code:-

```
# streamlit_app.py
import streamlit as st

st.title("PDF AI Assistant")
user_input = st.text_input("Ask a question from your notes:")

if user_input:
    answer = crew.run(input=user_input)
```

```
    st.write(answer)
```

5. (Optional) Replace LLM with LLaMA via Ollama

To use a local LLaMA model instead of OpenAI:

Python code:-

```
from langchain.llms import Ollama
llm = Ollama(model="llama3")
```

To **replace `ChatOpenAI` with `Ollama`** in your **CrewAI setup**, you need to do the following:

□ 1. Install Ollama and Pull LLaMA Model

If you haven't already:

```
# Install Ollama (Mac/Linux/Windows)
brew install ollama   # or use the installer from https://ollama.com
# Pull the LLaMA 3 model (7B recommended for local use)
ollama pull llama3
```

□ 2. Install LangChain Ollama Support

Make sure the LangChain version supports `Ollama` and install dependencies:

pip install langchain

🔁 3. Use Ollama in Place of ChatOpenAI

Replace this:

```
from langchain.chat_models import ChatOpenAI
llm = ChatOpenAI(model="gpt-4")
```

With this:

```
from langchain_community.llms import Ollama
llm = Ollama(model="llama3")
```

🧠 4. Updated CrewAI Setup with Ollama

Here's a **minimal working example** using `CrewAI` with **Ollama**:

```
from crewai import Agent, Task, Crew
from langchain_community.llms import Ollama

# Instantiate LLaMA 3 via Ollama
llm = Ollama(model="llama3")

# Define a simple tool (optional)
class DummyTool:
    def run(self, query):
        return f"Simulated document retrieval for: {query}"

tool = DummyTool()

# Define agents
researcher = Agent(
    role="Researcher",
    goal="Find relevant information from the notes",
    tools=[tool],
    llm=llm,
    verbose=True
)

responder = Agent(
    role="Answer Generator",
    goal="Use the info and give a clear answer",
```

```
    llm=llm,
    verbose=True
)

# Define task
task = Task(
    description="Answer the user's question using available tools and notes",
    agent=responder
)

# Define crew
crew = Crew(
    agents=[researcher, responder],
    tasks=[task],
    verbose=True
)

# Run
question = "What are the main points from the second chapter?"
result = crew.run(question)
print(result)
```

Important Notes

- Make sure **Ollama is running** (`ollama run llama3`) before starting the app.
- LLaMA via Ollama runs locally and might be slower than OpenAI if your system is limited on RAM or CPU.