Let's walk through a **step-by-step guide** for using **LangChain** with **Ollama** and **LLaMA 3** – a powerful **open-source LLM** from Meta that you can run locally.

□ What is Ollama?

Ollama is a tool to run large language models like **LLaMA 3**, **Mistral**, and others **locally on your machine**.

- Works offline
- Supports models like LLaMA3, Mistral, Gemma, etc.
- Can be easily integrated with LangChain

□ Prerequisites

□ 1. Install Ollama

Go to: https://ollama.com/download
Follow the instructions for **macOS**, **Linux**, or **Windows (WSL)**.

Then, run the model:

```
ollama run llama3
```

Or to download without running:

```
ollama pull llama3
```

□ LLaMA3 8B is the default. You can also use `llama3:70b` if your system can handle it.

□ 2. Set Up Python Environment

```
pip install langchain langchain-community
```

(We'll use `langchain-community` for Ollama integration.)

□ Step-by-Step LangChain + Ollama + LLaMA3 Guide

Step 1: Connect LangChain to Ollama

```
from langchain_community.llms import Ollama

llm = Ollama(model="llama3")
```

That's it! The Ollama server listens at `http://localhost:11434` by default.

Step 2: Ask LLaMA 3 a Question

```
response = llm.invoke("Explain quantum computing in simple terms.")
print(response)
```

Step 3: Use a Prompt Template

```
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

prompt = PromptTemplate.from_template(
    "What are 3 good names for a company that makes {product}?"
)

chain = LLMChain(llm=llm, prompt=prompt)

result = chain.run("eco-friendly packaging")
print(result)
```

Step 4: Add Memory to Conversation

```
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory()
conversation = ConversationChain(llm=llm, memory=memory, verbose=True)

print(conversation.run("Hello, I'm Alice."))
print(conversation.run("What did I just say my name was?"))
```

Step 5: Load a Local Text File and Ask Questions

```
from langchain.document_loaders import TextLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
```

```
from langchain.chains import RetrievalQA

# 1. Load a file
loader = TextLoader("example.txt")
docs = loader.load()

# 2. Split the document
splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=50)
splits = splitter.split_documents(docs)

# 3. Embeddings (use sentence-transformers, works well locally)
embedding = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

# 4. Create a vector store
db = FAISS.from_documents(splits, embedding)

# 5. Create retriever chain
retriever = db.as_retriever()
qa = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)

# 6. Ask questions
print(qa.run("What is the document about?"))
```

 `pip install sentence-transformers faiss-cpu` if needed.

Step 6: (Optional) Use Ollama with Agents

LangChain supports agents like `ReAct` using Ollama.

```
from langchain.agents import load_tools, initialize_agent
from langchain.agents.agent_types import AgentType

tools = load_tools(["llm-math"], llm=llm)

agent = initialize_agent(
    tools, llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True
)

agent.run("What is 17 * 23?")
```

□ Summary

Step	What You Did
1	Installed Ollama and pulled LLaMA 3
2	Connected LangChain to Ollama
3	Used prompt templates and chains
4	Added conversation memory

Step	What You Did
5	Loaded a local file and did RAG
6	(Optional) Used LangChain agents

□ Useful Commands

```
ollama run llama3          # Start the model
ollama pull llama3:70b     # Pull larger model
ollama list                # See available models
ollama serve               # Start Ollama server
```

□ Optional: Full Environment Setup

```
pip install langchain langchain-community sentence-transformers faiss-cpu
```

□ Full Python Code (LangChain + Ollama + LLaMA3 + Agent)

```
# main.py

from langchain_community.llms import Ollama
```

```
from langchain.agents import load_tools, initialize_agent
from langchain.agents.agent_types import AgentType

# Step 1: Load the local LLaMA 3 model via Ollama
llm = Ollama(model="llama3")

# Step 2: Load a tool — in this case, LLM-powered math calculator
tools = load_tools(["llm-math"], llm=llm)

# Step 3: Create a zero-shot reasoning agent
agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)

# Step 4: Run the agent with a math task
response = agent.run("What is 17 * 23?")
print("\nFinal Answer:", response)
```

let's expand the previous example to include **loading a PDF**, **indexing it with FAISS**, and **chatting with it using LangChain + LLaMA 3 (via Ollama)**.

□ What This Does

- Loads a **PDF** from your local machine
- Splits it into chunks
- Embeds the chunks with `sentence-transformers`
- Stores them in **FAISS**
- Uses an **agent with tools** to:
    - Answer questions about the PDF
    - Do reasoning (like math) using LLaMA 3

□ Required Packages

Install all dependencies:

```
pip install langchain langchain-community pypdf faiss-cpu sentence-
transformers sympy
```

□ Folder Structure

```
project/
├── my_document.pdf        ← Your PDF file
└── main.py                ← The code below
```

□ Full Code (Chat with PDF + Math Agent)

```
# main.py

from langchain_community.llms import Ollama
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.chains import RetrievalQA
from langchain.agents import load_tools, initialize_agent, Tool
from langchain.agents.agent_types import AgentType

# STEP 1: Load LLaMA 3 (via Ollama)
llm = Ollama(model="llama3")

# STEP 2: Load your PDF
loader = PyPDFLoader("my_document.pdf")
documents = loader.load()

# STEP 3: Split into chunks
splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=50)
```

```
chunks = splitter.split_documents(documents)

# STEP 4: Embed using sentence-transformers
embedding = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

# STEP 5: Store in FAISS
vectorstore = FAISS.from_documents(chunks, embedding)

# STEP 6: Create a retriever for Q&A
retriever = vectorstore.as_retriever()

qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)

# STEP 7: Wrap it as a Tool
pdf_tool = Tool(
    name="PDF QA Tool",
    func=qa_chain.run,
    description="Use this to answer questions from the PDF document."
)

# STEP 8: Add built-in tools (e.g. LLM Math)
tools = load_tools(["llm-math"], llm=llm)

# Combine tools
tools.append(pdf_tool)

# STEP 9: Initialize Agent with tools
agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)

# STEP 10: Ask it something about the PDF (or math)
query = "What are the key findings in the document?"
response = agent.run(query)

print("\nAnswer:", response)
```

□ Example Questions to Try

- “Summarize the document.”
- “What are the challenges mentioned?”
- “Who wrote this paper?”
- “What is 88 * 13?” ← math tool will solve this

□ Notes

- □ **Local embeddings** with `sentence-transformers` ensure everything can run **offline**.
- □ You can replace `HuggingFaceEmbeddings` with OpenAI or others if online.
- □ You can extend to multiple PDFs — just load and concatenate them before splitting.

□ Summary

Step	Component
1	Load and split your PDF
2	Embed chunks and store in FAISS
3	Use `RetrievalQA` to query PDF
4	Combine with LangChain Agent + Tools
5	Ask questions and get smart answers

Let's turn your `main.py` into a **Gradio-powered interface** where users can:

□ Upload a **PDF**
□ Ask **questions** about it (or general questions)
□ Get **answers from LLaMA 3** with tool support (like math)

□ Updated Full Code with Gradio Interface

```
# main.py

import gradio as gr
from langchain_community.llms import Ollama
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.chains import RetrievalQA
from langchain.agents import load_tools, initialize_agent, Tool
from langchain.agents.agent_types import AgentType
import tempfile
import os

# GLOBAL OBJECTS (will be set once a PDF is uploaded)
agent = None
vectorstore = None

# Step A: Setup LLaMA 3 model via Ollama
llm = Ollama(model="llama3")


# Step B: Function to handle PDF upload and initialize agent
def setup_agent_from_pdf(pdf_file):
    global agent, vectorstore

    # Save uploaded file to a temporary location
    with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as
tmp_file:
        tmp_file.write(pdf_file.read())
        pdf_path = tmp_file.name

    # Load and split PDF
    loader = PyPDFLoader(pdf_path)
    documents = loader.load()
    splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=50)
    chunks = splitter.split_documents(documents)

    # Embeddings + FAISS
    embedding = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
    vectorstore = FAISS.from_documents(chunks, embedding)
    retriever = vectorstore.as_retriever()

    # QA chain
    qa_chain = RetrievalQA.from_chain_type(
        llm=llm,
        retriever=retriever,
        return_source_documents=True
    )

    # Tools
    pdf_tool = Tool(
        name="PDF QA Tool",
        func=qa_chain.run,
```

```
        description="Use this to answer questions from the PDF document."
    )
    tools = load_tools(["llm-math"], llm=llm)
    tools.append(pdf_tool)

    # Agent
    agent = initialize_agent(
        tools=tools,
        llm=llm,
        agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
        verbose=True
    )

    return "□ PDF loaded and ready to chat!"


# Step C: Function to handle queries
def chat_with_agent(question):
    if agent is None:
        return "□ Please upload a PDF first."
    return agent.run(question)


# Step D: Gradio Interface
with gr.Blocks() as demo:
    gr.Markdown("## □ Chat with PDF using LLaMA 3 + LangChain + Ollama")

    with gr.Row():
        pdf_file = gr.File(label="□ Upload your PDF", file_types=[".pdf"])
        upload_btn = gr.Button("Load PDF")

    status = gr.Textbox(label="Status", interactive=False)

    question = gr.Textbox(label="Ask a question about the PDF or general
math")
    answer = gr.Textbox(label="Answer", interactive=False)

    upload_btn.click(fn=setup_agent_from_pdf, inputs=[pdf_file],
outputs=[status])
    question.submit(fn=chat_with_agent, inputs=[question], outputs=[answer])

demo.launch()
```

□ Requirements

Install dependencies:

```
pip install gradio langchain langchain-community pypdf sentence-transformers
faiss-cpu sympy
```

Also install Ollama and pull LLaMA 3:

```
ollama pull llama3
```

And keep Ollama running:

```
ollama run llama3
```