? What Is LangChain?

LangChain is a Python (and JavaScript) framework for developing **applications powered by language models**. It simplifies building complex LLM workflows like:

- AI agents that can reason and act
- Chatbots with memory
- LLMs that interact with tools (like search, Python, APIs, etc.)
- Document Q&A systems

□ Basic Components of LangChain

Before building the agent, you need to understand the core LangChain components:

Component	Description
LLM	The language model (like OpenAI's GPT-4)
PromptTemplate	Custom prompt to guide LLM's output
Tools	External utilities the agent can use (search, calculator, etc.)
Agent	The decision-making entity using the LLM and tools
Memory	Stores past interactions or context
Chain	A sequence of calls (LLMs + tools) structured logically

---

□ Step-by-Step: Build an AI Agent with LangChain

□ 1. Install LangChain and Dependencies

```
pip install langchain openai
```

Optional (for tools):

```
pip install duckduckgo-search
pip install wikipedia
```

□ 2. Set Up Your LLM

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(model_name="gpt-4", temperature=0.7)
```

Make sure you've set your OpenAI API key:

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

□ 3. Define Some Tools

LangChain has built-in tools like a **calculator**, **search**, and **Python REPL**.

```
from langchain.agents import load_tools

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

You can also create **custom tools** using functions.

□ 4. Create the Agent

Use LangChain's `initialize_agent` to wire up everything:

```
from langchain.agents import initialize_agent, AgentType

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

Agent Types:

- `ZERO_SHOT_REACT_DESCRIPTION`: No examples, just reasoning with descriptions
- `CHAT_ZERO_SHOT_REACT_DESCRIPTION`: Chat-based
- `STRUCTURED_CHAT_ZERO_SHOT_REACT_DESCRIPTION`: Supports structured tools

□ 5. Ask the Agent a Question

```
response = agent.run("Who is the current president of France, and what is 456 * 78?")
print(response)
```

□ How It Works

The agent:

1. Reads your input
2. Decides what tool to use (e.g., Wikipedia, math)
3. Uses the tool
4. Combines results
5. Responds intelligently

□ Bonus: Custom Tool Example

```
from langchain.agents import Tool

def get_weather(location: str) -> str:
    return f"The weather in {location} is sunny."

weather_tool = Tool(
    name="get_weather",
    func=get_weather,
    description="Use this tool to get the weather of a location."
)

agent = initialize_agent(
    tools + [weather_tool],
    llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)
```

Now your agent can use your custom `get_weather` tool.

□ Add Memory (Optional)

To make your agent **remember conversations**, you can use memory:

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

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

response = conversation.predict(input="Hi, I'm Sarah.")
print(response)
```

□ Summary

Step	What You Did
1	Installed LangChain and OpenAI
2	Set up an LLM
3	Loaded tools
4	Initialized an agent
5	Asked questions and got dynamic results
6	(Optional) Added memory or custom tools

□ Learn More

- LangChain Docs
- Agents Overview
- Toolkits
- Memory

Here's a **full working example** of how to create a simple **LangChain AI agent** using the **LLaMA 3 model from Ollama** instead of OpenAI.

This guide assumes:

- You have **Ollama installed**: https://ollama.com
- You've already pulled the model:
- `ollama pull llama3`

□ Step-by-Step: LangChain Agent with Ollama (LLaMA 3)

□ 1. Install Required Libraries

Make sure you have all the dependencies:

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

□ 2. Python Code (Full Agent with Tools using LLaMA 3)

```
# llama3_agent.py

from langchain.agents import initialize_agent, AgentType, Tool
from langchain.agents.agent_toolkits import load_tools
from langchain_community.llms import Ollama
from langchain.agents import load_tools

# Step 1: Load LLaMA 3 via Ollama
llm = Ollama(model="llama3", temperature=0.7)

# Step 2: Load built-in tools
tools = load_tools(["llm-math"], llm=llm)  # includes calculator

# Step 3: Optional - Add a custom tool
def get_weather(location: str) -> str:
    return f"The weather in {location} is sunny with 25°C."

weather_tool = Tool(
    name="get_weather",
    func=get_weather,
    description="Useful to get current weather for a location. Input should
be a city name."
)

# Step 4: Initialize the agent with tools + LLM
agent = initialize_agent(
    tools + [weather_tool],
    llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)

# Step 5: Run agent
if __name__ == "__main__":
    question = "What is 345 * 23? And what's the weather in Tokyo?"
    response = agent.run(question)
    print("\nAgent Response:\n", response)
```

▶□ 3. Run the Code

Save the file as `llama3_agent.py`, then run:

```
python llama3_agent.py
```

□ Expected Output (sample)

The output will look something like this (abbreviated):

```
> Entering new AgentExecutor chain...
```

```
Thought: I need to calculate the multiplication and check the weather
Action: Calculator
Action Input: 345 * 23
Observation: 7935
Thought: Now I'll get the weather in Tokyo
Action: get_weather
Action Input: Tokyo
Observation: The weather in Tokyo is sunny with 25°C.
Final Answer: 345 * 23 is 7935. The weather in Tokyo is sunny with 25°C.

Agent Response:
 345 * 23 is 7935. The weather in Tokyo is sunny with 25°C.
```

□ Summary

You now have a LangChain AI Agent using:-

Component	Details
LLM	LLaMA 3 via Ollama
Tools	Math + Custom Weather
Agent Type	`ZERO_SHOT_REACT_DESCRIPTION`
Framework	LangChain

We'll enhance your **LangChain agent** with:

1. □ **Memory** — so the agent remembers the conversation
2. □ **Chaining** — to create structured workflows (e.g., pre-processing inputs or post-processing outputs)
3. □ **ChatPromptTemplate** — for flexible prompting

□ Updated Plan

We'll create a conversational AI agent using **LLaMA 3 (Ollama)** with:

Feature	Description
LLM	Ollama running `llama3`
Tools	Math + Weather (custom)
Memory	Stores chat history
Prompt	Custom chat prompt
Chain	Combines LLM + memory in a structured workflow

□ Full Python Script: `llama3_chat_agent.py`

```
# llama3_chat_agent.py

from langchain_community.llms import Ollama
from langchain.agents import Tool, initialize_agent, AgentType
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationChain
from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.messages import SystemMessage
from langchain_core.runnables import Runnable
from langchain_core.output_parsers import StrOutputParser
from langchain.schema.runnable import RunnableMap
from langchain.chains import LLMChain
from langchain.agents.agent_toolkits import load_tools

# Step 1: Load LLaMA 3 from Ollama
llm = Ollama(model="llama3", temperature=0.7)

# Step 2: Add tools
def get_weather(location: str) -> str:
    return f"The weather in {location} is sunny with 25°C."

weather_tool = Tool(
    name="get_weather",
    func=get_weather,
    description="Use to get current weather in a location. Input should be a
city name."
)

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

# Step 3: Setup memory
```

```
memory = ConversationBufferMemory(memory_key="chat_history",
return_messages=True)

# Step 4: Create ChatPromptTemplate
prompt = ChatPromptTemplate.from_messages([
    SystemMessage(content="You are a helpful AI assistant. Use tools when
needed."),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{input}")
])

# Step 5: Build a Chain (LLM + Prompt + Memory)
chain = LLMChain(
    llm=llm,
    prompt=prompt,
    memory=memory,
    verbose=True
)

# Step 6: Initialize Agent (with memory and tools)
agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,
    memory=memory,
    verbose=True
)

# Step 7: Run the Agent interactively
def main():
    print("□ LangChain Agent (LLaMA 3 + Memory + Prompt + Tools)")
    print("Type 'exit' to stop.")

    while True:
        user_input = input("\nYou: ")
        if user_input.lower() in ["exit", "quit"]:
            break
        response = agent.run(user_input)
        print(f"\n□ Agent: {response}")

if __name__ == "__main__":
    main()
```

□ How to Run

1. Save the script as `llama3_chat_agent.py`
2. Run it:

```
python llama3_chat_agent.py
```

3. Try a conversation:

```
You: What is 67 * 89?
```

```
 Agent: 67 * 89 is 5963.

You: What's the weather in Paris?
 Agent: The weather in Paris is sunny with 25°C.

You: What did I ask you first?
 Agent: You asked me to calculate 67 * 89.
```

Features Recap

Feature	Done
LLaMA 3 via Ollama	
Custom Tool (Weather)	
Built-in Tool (Math)	
Memory (ConversationBufferMemory)	
ChatPromptTemplate	
Chain (LLMChain)	
Interactive chat loop	

Here's a **very simple LangChain agent** that uses the **math tool** and **LLaMA 3 via Ollama** to answer your math questions.

□ Goal

- Load **LLaMA 3 (Ollama)**
- Use only the **math tool**
- Let the agent answer **math questions**
- No memory, no custom prompt — just **clean and focused**

□ 1. Install Required Packages

Make sure you have these installed:

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

Final Python Code: `simple_math_agent_with_memory.py`

```
# simple_math_agent_with_memory.py

from langchain_community.llms import Ollama
from langchain.agents import initialize_agent, AgentType
from langchain.agents.agent_toolkits import load_tools
from langchain.memory import ConversationBufferMemory

# Step 1: Load LLaMA 3 model from Ollama
llm = Ollama(model="llama3", temperature=0.0)

# Step 2: Load only the math tool
tools = load_tools(["llm-math"], llm=llm)

# Step 3: Set up memory to store the chat history
memory = ConversationBufferMemory(memory_key="chat_history",
return_messages=True)

# Step 4: Initialize the agent with memory
agent = initialize_agent(
    tools=tools,
    llm=llm,
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,  # Needed for memory
    memory=memory,
    verbose=True
)
```

```
# Step 5: Run the agent in a loop
if __name__ == "__main__":
    print("□ Math Agent with Memory (LLaMA 3 + Math Tool + Chat History)\n")
    while True:
        question = input("You: ")
        if question.lower() in ["exit", "quit"]:
            break
        response = agent.run(question)
        print(f"□ Agent: {response}\n")
```

▶□ Example Interaction

```
You: What is 123 * 4?
□ Agent: 123 * 4 = 492

You: What was the number you just gave me plus 8?
□ Agent: 492 + 8 = 500
```

□ Recap of Features

Feature	□ Status
LLaMA 3 (Ollama)	□
Math Tool	□
Conversation Memory	□
Interactive Chat	□
Simple + Clean	□

Here you go! Below is your updated **Simple Math Agent** with **Gradio UI**, using **LangChain + LLaMA 3 via Ollama** + **Math Tool**.

□ What This Does

- Uses **LLaMA 3** (via Ollama)
- Supports **math questions**
- Built with **Gradio** UI
- Runs in the browser □

□ First: Install Required Packages

Make sure you have the following installed:

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

And make sure you've pulled the model:

```
ollama pull llama3
```

□ Full Code: `simple_math_agent_gradio.py`

```
# simple_math_agent_gradio.py

import gradio as gr
from langchain_community.llms import Ollama
from langchain.agents import initialize_agent, AgentType
from langchain.agents.agent_toolkits import load_tools
from langchain.memory import ConversationBufferMemory

# Load LLaMA 3 model from Ollama
llm = Ollama(model="llama3", temperature=0.0)

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

# Add memory for chat context
memory = ConversationBufferMemory(memory_key="chat_history",
return_messages=True)

# Initialize agent with chat memory
agent = initialize_agent(
    tools=tools,
```

```
    llm=llm,
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,
    memory=memory,
    verbose=False  # Set to True for detailed logs
)

# Function for Gradio interface
def chat_with_agent(user_input, chat_history=[]):
    response = agent.run(user_input)
    chat_history.append((user_input, response))
    return chat_history, chat_history

# Gradio UI setup
chat_interface = gr.ChatInterface(
    fn=chat_with_agent,
    title="□ Math Agent (LLaMA 3)",
    description="Ask math questions like 'What is 23 * 47?' or 'Add 20 to the
previous answer.'",
    theme="default",
    examples=["What is 15 * 12?", "What is the square root of 256?", "Add 10
to the previous result."],
)

# Launch the app
if __name__ == "__main__":
    chat_interface.launch()
```

▶□ How to Run

```
python simple_math_agent_gradio.py
```

Then visit the **local URL** Gradio gives you, like:

```
Running on local URL: http://127.0.0.1:7860
```

□ Features Recap

Feature	□ Included
LLaMA 3 (Ollama)	□
Math Tool	□
Gradio Chat UI	□
Memory (context-aware)	□
Lightweight	□

Let me know if you'd like to:

- Add more tools (e.g. weather, Wikipedia)
- Enable voice input
- Add PDF/doc upload
- Deploy this to Hugging Face Spaces or locally as a web app