Here's a **full Python example** showing how to use **LangChain with Ollama** to run a conversation with a **LLaMA model** locally:-

---

Prerequisites

1. Install Ollama and download a model (e.g. `llama2`):

```
ollama pull llama2
```

2. Install Python packages:

```
pip install langchain ollama
```

Example Python script: `ollama_langchain_example.py`

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

def main():
    # Initialize the Ollama LLM with your chosen model
    llm = Ollama(model="llama2")

    # Setup conversation memory to keep context
    memory = ConversationBufferMemory(memory_key="chat_history")

    # Create a conversation chain with the LLM and memory
    conversation = ConversationChain(llm=llm, memory=memory)

    print("□ Ollama + LangChain Chatbot (type 'exit' to quit)")

    while True:
        user_input = input("\nYou: ")
        if user_input.strip().lower() == "exit":
            break

        # Get the AI's response, maintaining conversation history
        response = conversation.predict(input=user_input)

        print(f"Bot: {response}")

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

How it works:

- `Ollama(model="llama2")` connects to your local Ollama LLaMA model.
- `ConversationBufferMemory` keeps the dialogue context.
- `ConversationChain` ties the LLM and memory together for smooth multi-turn chat.
- The script runs a REPL loop you can chat with, exiting on `"exit"`.

Run it:

```
python ollama_langchain_example.py
```

Complete script with Gradio :-

```
import gradio as gr
from langchain.llms import Ollama
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferMemory

# Initialize Ollama LLM once (to avoid reloading every call)
llm = Ollama(model="llama2")
memory = ConversationBufferMemory(memory_key="chat_history")
conversation = ConversationChain(llm=llm, memory=memory)

def chat(user_input):
    if user_input.strip().lower() == "exit":
        return "Goodbye!", None
    response = conversation.predict(input=user_input)
    return response, conversation.memory.chat_memory.messages

with gr.Blocks() as demo:
    chatbot = gr.Chatbot()
    msg = gr.Textbox(placeholder="Ask me something...")
    clear = gr.Button("Clear")

    def respond(message, chat_history):
        bot_response, _ = chat(message)
        chat_history = chat_history or []
        chat_history.append((message, bot_response))
        return "", chat_history

    msg.submit(respond, [msg, chatbot], [msg, chatbot])
    clear.click(lambda: (None, []), None, chatbot)

demo.launch()
```

How it works:

- The conversation chain and memory are initialized **once globally** so context is preserved between turns.
- The `chat` function takes the user input and returns a response from the LLM, updating the conversation memory.
- The Gradio interface uses a `Chatbot` component to display conversation history.
- The user types input and presses enter to send, updating the chat window.
- A **Clear** button resets the conversation.