Analytics Vidhya

Build a LLM-powered Chatbot with LangGraph

Dipanjan Sarkar

Head of Community & Principal AI Scientist at Analytics Vidhya

Google Developer Expert - ML & Cloud Champion Innovator

Published Author

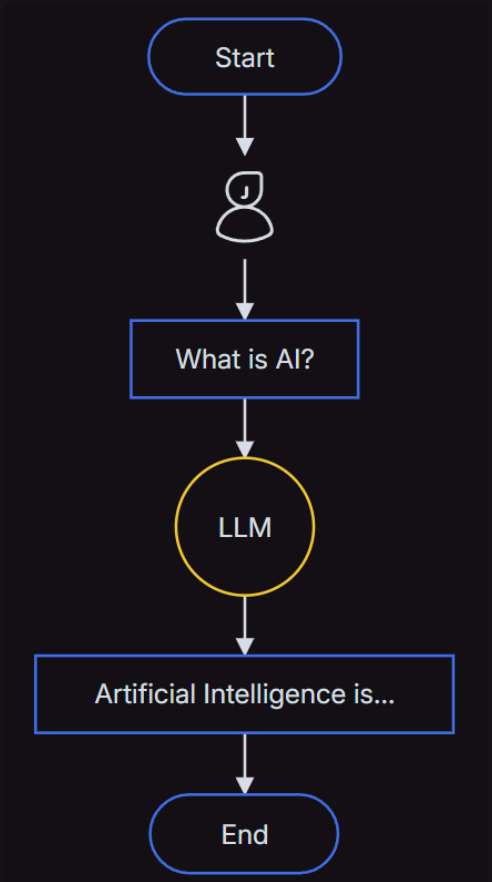
LangGraph isn't just for static graphs—it powers stateful, agentic applications with LLMs.

Let’s build a simple LLM-powered chatbot using LangGraph to respond directly to user messages.

LLM-powered Chatbot in LangGraph

Key Features

- Graph-based workflow coordination.
- Use of reducers for state management.
- Integration with LLMs for generating responses.



Analytics Vidhya

LLM-powered Chatbot in LangGraph

State Management

- Define a State schema using TypedDict.
- Use add_messages to append new messages to the agent state graph.

```python
from typing import Annotated
from typing_extensions import TypedDict
from langgraph.graph.message import add_messages
from langgraph.graph import StateGraph, START, END
from langchain_openai import ChatOpenAI

# create state schema
class State(TypedDict):
    messages: Annotated[list, add_messages]

# create chatbot node function
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

def chatbot(state: State):
    return {"messages": [llm.invoke(state["messages"])]}

# create graph
graph_builder = StateGraph(State)
graph_builder.add_node("chatbot", chatbot)
graph_builder.add_edge(START, "chatbot")
graph_builder.add_edge("chatbot", END)
graph = graph_builder.compile()

# execute state graph
response = graph.invoke({"messages": "Explain AI in 1 line to
a child"})
print(response['messages'][-1].content)

## OUTPUT
AI is like a smart robot that can learn and help us solve
problems or answer questions!
```

LLM-powered Chatbot in LangGraph

Node Functions

- Nodes represent functional units of work.
- Example: A chatbot node that processes input and generates LLM responses.

```python
from typing import Annotated
from typing_extensions import TypedDict
from langgraph.graph.message import add_messages
from langgraph.graph import StateGraph, START, END
from langchain_openai import ChatOpenAI

# create state schema
class State(TypedDict):
    messages: Annotated[list, add_messages]

# create chatbot node function
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

def chatbot(state: State):
    return {"messages": [llm.invoke(state["messages"])]}

# create graph
graph_builder = StateGraph(State)
graph_builder.add_node("chatbot", chatbot)
graph_builder.add_edge(START, "chatbot")
graph_builder.add_edge("chatbot", END)
graph = graph_builder.compile()

# execute state graph
response = graph.invoke({"messages": "Explain AI in 1 line to
a child"})
print(response['messages'][-1].content)

## OUTPUT
AI is like a smart robot that can learn and help us solve
problems or answer questions!
```

LLM-powered Chatbot in LangGraph

Graph Edges

- Specify entry (START) and exit (END) points for workflows.
- Use edges to define transitions between nodes.

```python
from typing import Annotated
from typing_extensions import TypedDict
from langgraph.graph.message import add_messages
from langgraph.graph import StateGraph, START, END
from langchain_openai import ChatOpenAI

# create state schema
class State(TypedDict):
    messages: Annotated[list, add_messages]

# create chatbot node function
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

def chatbot(state: State):
    return {"messages": [llm.invoke(state["messages"])]}

# create graph
graph_builder = StateGraph(State)
graph_builder.add_node("chatbot", chatbot)
graph_builder.add_edge(START, "chatbot")
graph_builder.add_edge("chatbot", END)
graph = graph_builder.compile()

# execute state graph
response = graph.invoke({"messages": "Explain AI in 1 line to
a child"})
print(response['messages'][-1].content)

## OUTPUT
AI is like a smart robot that can learn and help us solve
problems or answer questions!
```

Thanks