Building graphs for chatbots

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

datacamp

Chatbots with LangGraph

- **Custom agents with LangGraph**
  - **Workflow management**
    - Graph states
    - Agent states
  - **Chatbot construction**
    - Nodes
    - Edges

Graphs and agent states

Graph State

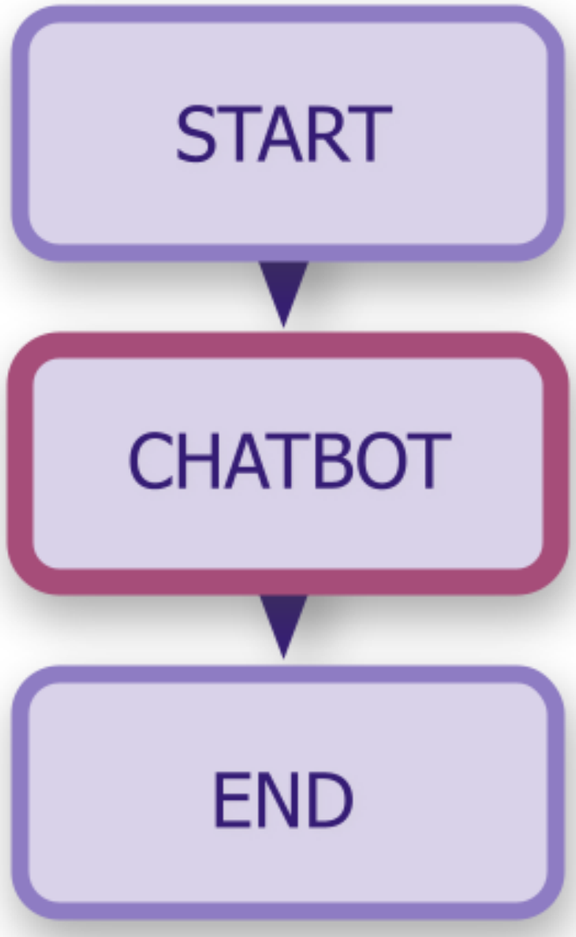
- Organizes different tasks
  - Tool use
  - LLM calls
- Order of tasks = **workflow**

Agent State

- Tracks agent's progress
- Logs task completion

Building an agent with LangGraph

Nodes and edges



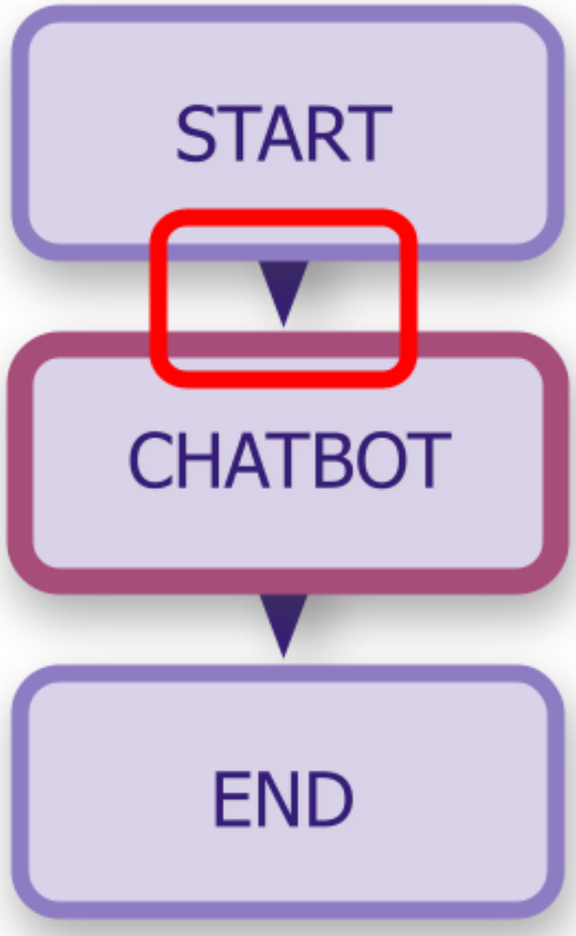
Nodes

- Functions or actions
  - Response
  - Tool call

Edges

- Rules connecting nodes

Nodes and edges



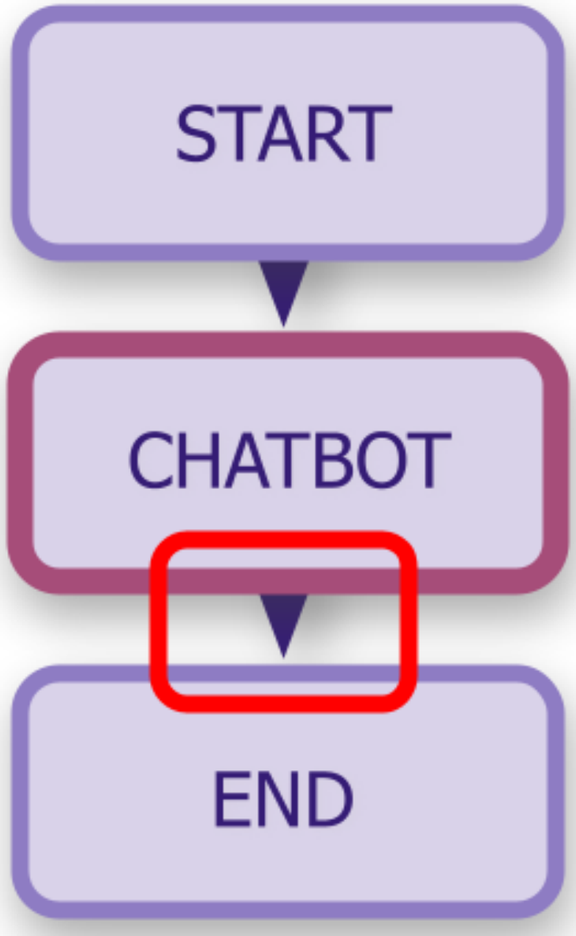
Nodes

- Tasks or actions
  - Response
  - Tool call

Edges

- Rules connecting nodes

Nodes and edges



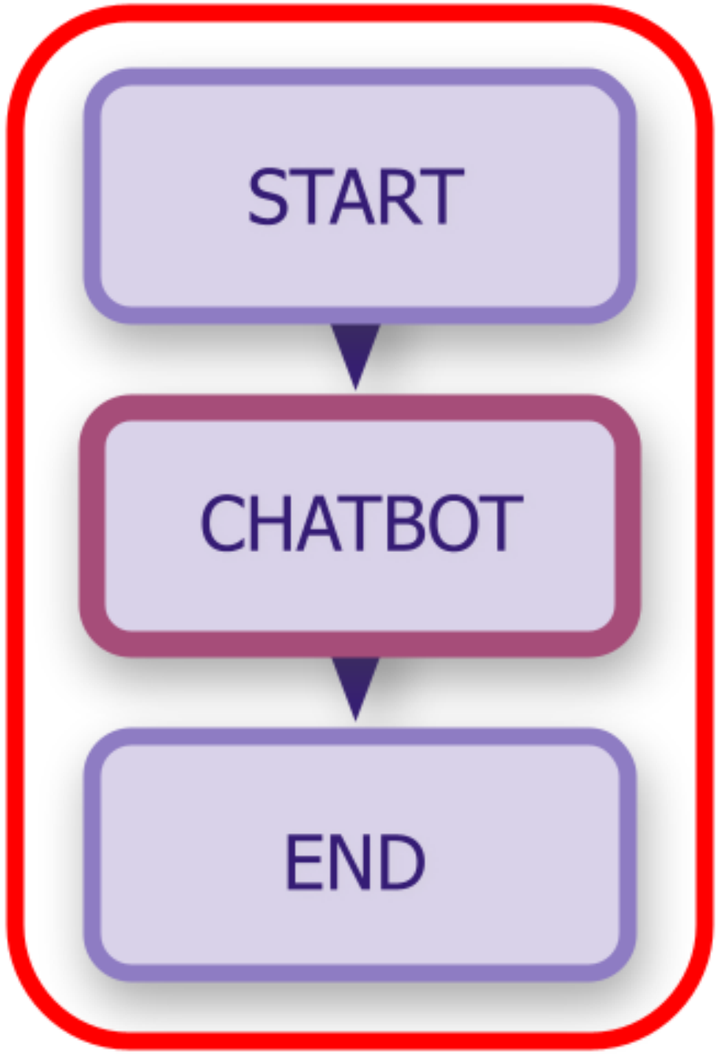
Nodes

- Tasks or actions
  - Response
  - Tool call

Edges

- Rules connecting nodes

Nodes and edges



Nodes

- Tasks or actions
  - Response
  - Tool call

Edges

- Rules connecting nodes

Pre-built Nodes

START and **END** nodes from LangGraph

Building graph and agent states

```
# Modules for structuring text
from typing import Annotated
from typing_extensions import TypedDict

# LangGraph modules for defining graphs
from langgraph.graph import StateGraph, START, END
from langgraph.graph.message import add_messages

# Module for setting up OpenAI
from langchain_openai import ChatOpenAI
```

Building graph and agent states

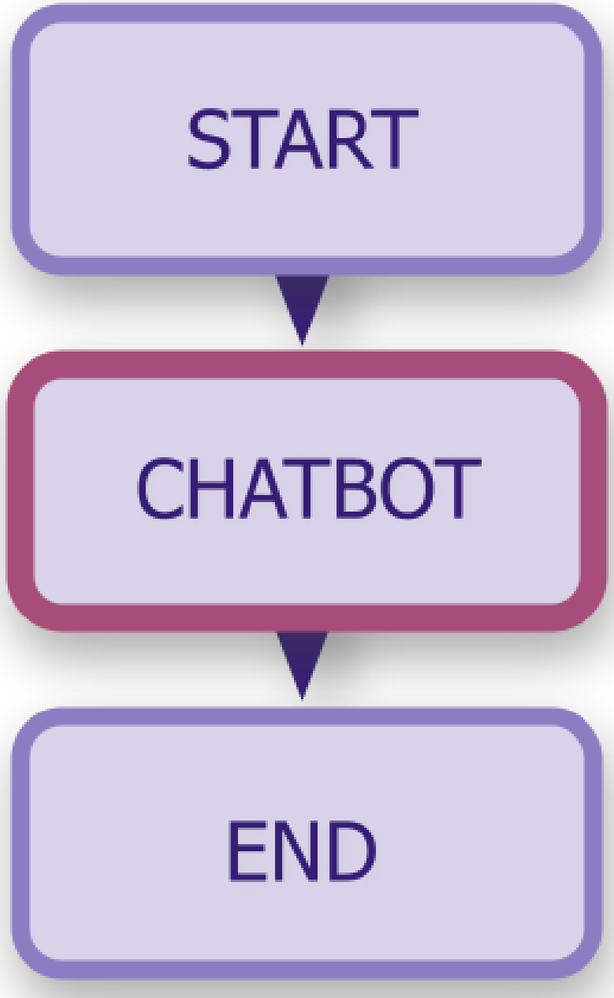
```
# Define the llm
llm = ChatOpenAI(model="gpt-4o-mini", api_key="OPENAI_API_KEY")

# Define the State
class State(TypedDict):

    # Define messages with metadata
    messages: Annotated[list, add_messages]

# Initialize StateGraph
graph_builder = StateGraph(State)
```

Adding nodes and edges

```
# Define chatbot function to respond
# with the model
def chatbot(state: State):
    return {"messages":
    [llm.invoke(state["messages"])]}

# Add chatbot node to the graph
graph_builder.add_node("chatbot",
                       chatbot)
```

Adding nodes and edges

```
# Define the start and end of the
# conversation flow
graph_builder.add_edge(START, "chatbot")
graph_builder.add_edge("chatbot", END)

# Compile the graph to prepare for
# execution
graph = graph_builder.compile()
```



Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Generating chatbot responses

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

datacamp

Streaming graph events

- Stream events in real-time
- Each event is a workflow step
- Track responses and tool calls
- Track chatbot progress

```
User: Where is the Congo?
Agent: Let me check some details
       for you...
Agent: AIMessage([Tool Call to
       Google Maps] Searching for
       information on "the Congo")
Agent: The Democratic Republic of
       Congo is a country in
       Central Africa.
```

Streaming LLM responses

```
# Define a function to execute the chatbot based on user input
def stream_graph_updates(user_input: str):

    # Start streaming events from the graph with the user's input
    for event in graph.stream({"messages": [("user", user_input)]}):

        # Retrieve and print the chatbot node responses
        for value in event.values():
            print("Agent:", value["messages"])

# Define the user query and run the chatbot
user_query = "Who is Mary Shelley?"
stream_graph_updates(user_query)
```

Streaming LLM responses

```
Agent: [AIMessage(content='Mary Shelley (1797-1851) was an English
novelist...
...best known for her groundbreaking work in the Gothic genre,
particularly for her novel "Frankenstein; or, The Modern Prometheus,"
published in 1818. This novel is often considered one of the earliest
examples of science fiction and explores themes of creation,
responsibility, and the nature of humanity through the story of Victor
Frankenstein, a scientist who creates a sentient creature in an
unorthodox experiment...',
response_metadata={'finish_reason': 'stop', 'model_name':
'gpt-4o-mini-...')]
```

LLMs and hallucinations

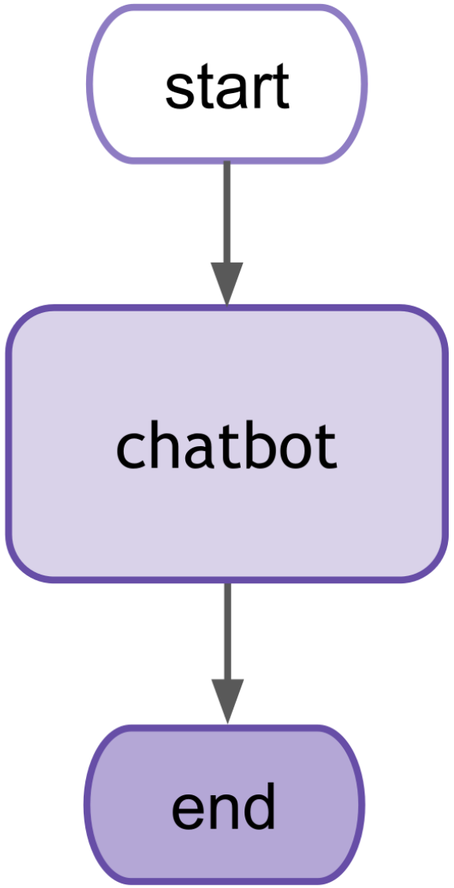
```
Agent: [AIMessage(content='Judith
Love Cohen was an American aerospace
engineer, and worked on various
space missions, including the Apollo
program... Cohen was also the mother
of actor and writer Adam Cohen,
who has spoken about her influence on
his life and career.
additional_kwargs={},
response_metadata={'finish_reason':
'stop', 'model_name':
'gpt-4o-mini-...)
```

Example hallucination

- Judith Love Cohen's famous son is Jack Black, not "Adam Cohen".

Generate a LangGraph diagram

```
# Import modules for chatbot diagram
from IPython.display import
Image, display

# Try generating and displaying
# the graph diagram
try:
    display(Image(graph.get_graph()
    .draw_mermaid_png()))

# Return an exception if necessary
except Exception:
    print("Additional dependencies
    required.")
```



Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

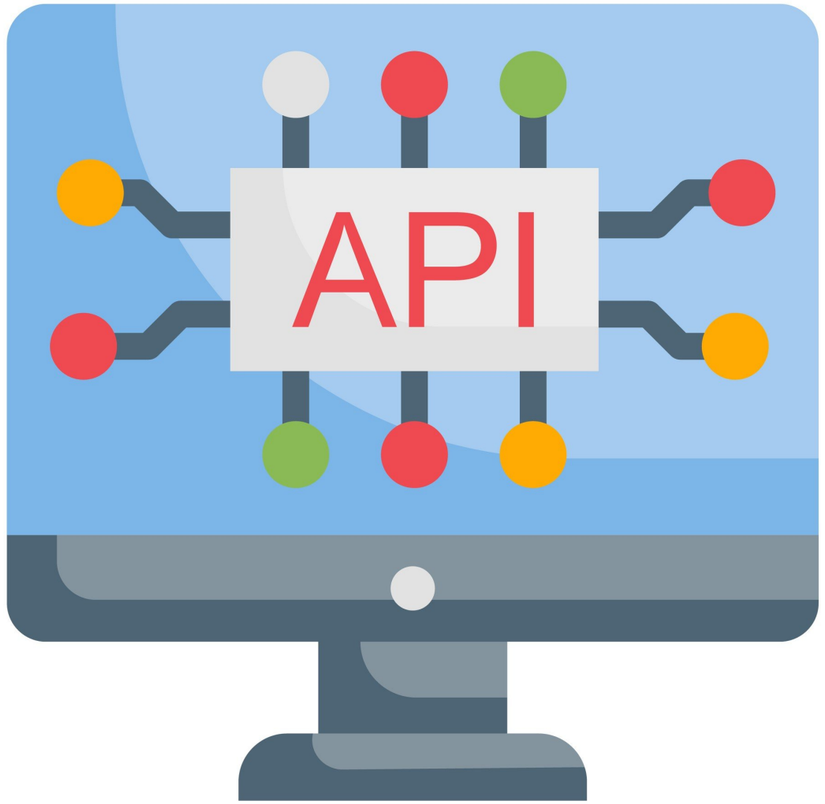
Adding external tools to a chatbot

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

datacamp

Externals tools with LangGraph



- API tools for chatbots
  - News
  - Databases
  - Social media
  - Etc.

Adding a Wikipedia tool

Adding a Wikipedia tool

```
# Modules for building a Wikipedia tool
from langchain_community.utilities import WikipediaAPIWrapper
from langchain_community.tools import WikipediaQueryRun

# Initialize Wikipedia API wrapper to fetch top 1 result
api_wrapper = WikipediaAPIWrapper(top_k_results=1)

# Create a Wikipedia query tool using the API wrapper
wikipedia_tool = WikipediaQueryRun(api_wrapper=api_wrapper)
tools = [wikipedia_tool]
```

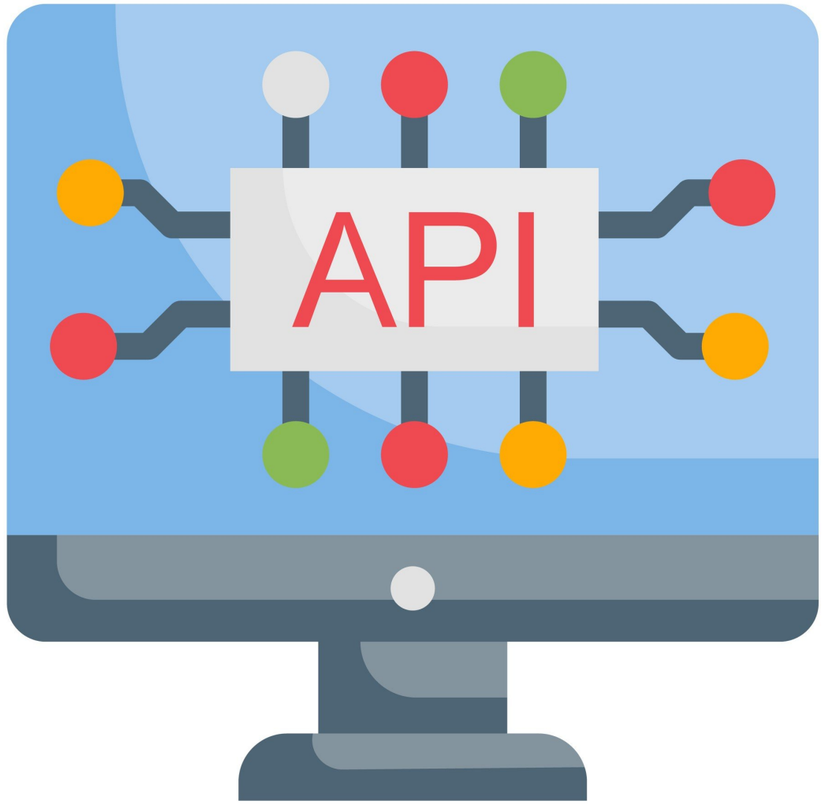
Adding a Wikipedia tool

```
# Bind the Wikipedia tool to
# the language model
llm_with_tools = llm.bind_tools(tools)


# Modify chatbot function to
# respond with Wikipedia
def chatbot(state: State):
    return {"messages":
    [llm_with_tools.invoke(
    state["messages"])]}
```

- Bind tools

- Update chatbot node
- Enable Wikipedia
- LLM decides tool calls

Other API tools



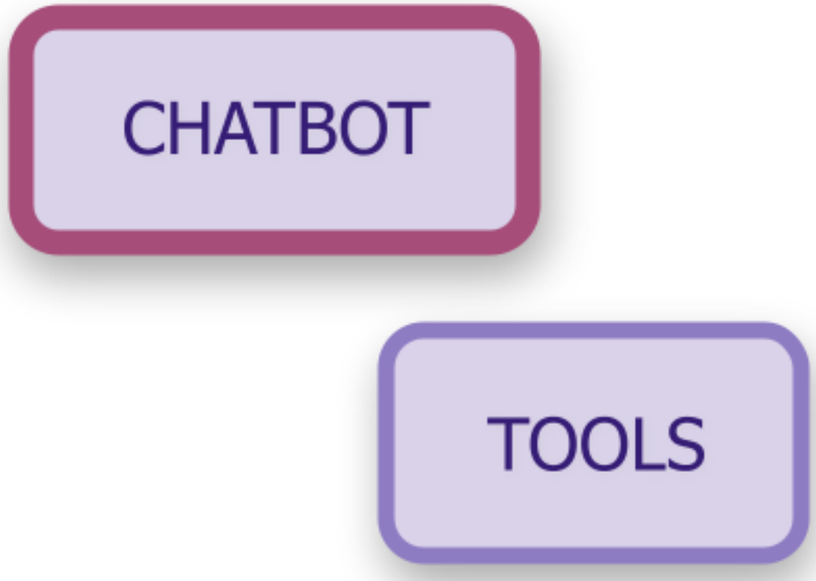
- LangChain API documentation

Adding tool nodes

```
# Modules for adding tool conditions
# and nodes
from langgraph.prebuilt import
ToolNode, tools_condition

# Add chatbot node to the graph
graph_builder.add_node("chatbot",
                       chatbot)
```

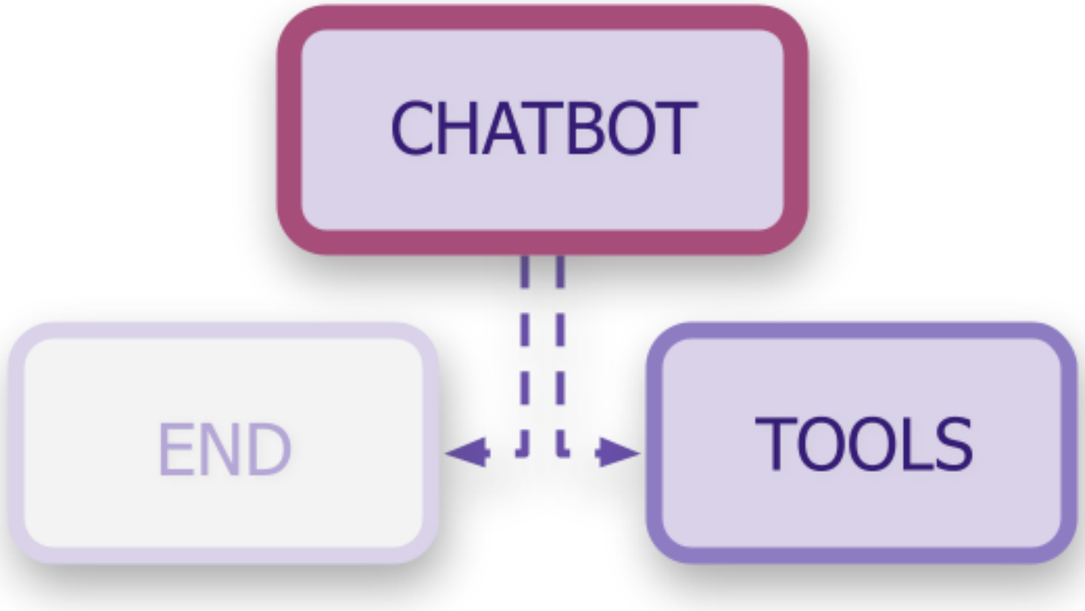
Adding tool nodes

```
# Modules for adding tool conditions
# and nodes
from langgraph.prebuilt import
ToolNode, tools_condition

# Add chatbot node to the graph
graph_builder.add_node("chatbot",
                       chatbot)

# Create a ToolNode to handle tool calls
# and add it to the graph
tool_node = ToolNode(tools=[wikipedia_tool])
graph_builder.add_node("tools", tool_node)
```



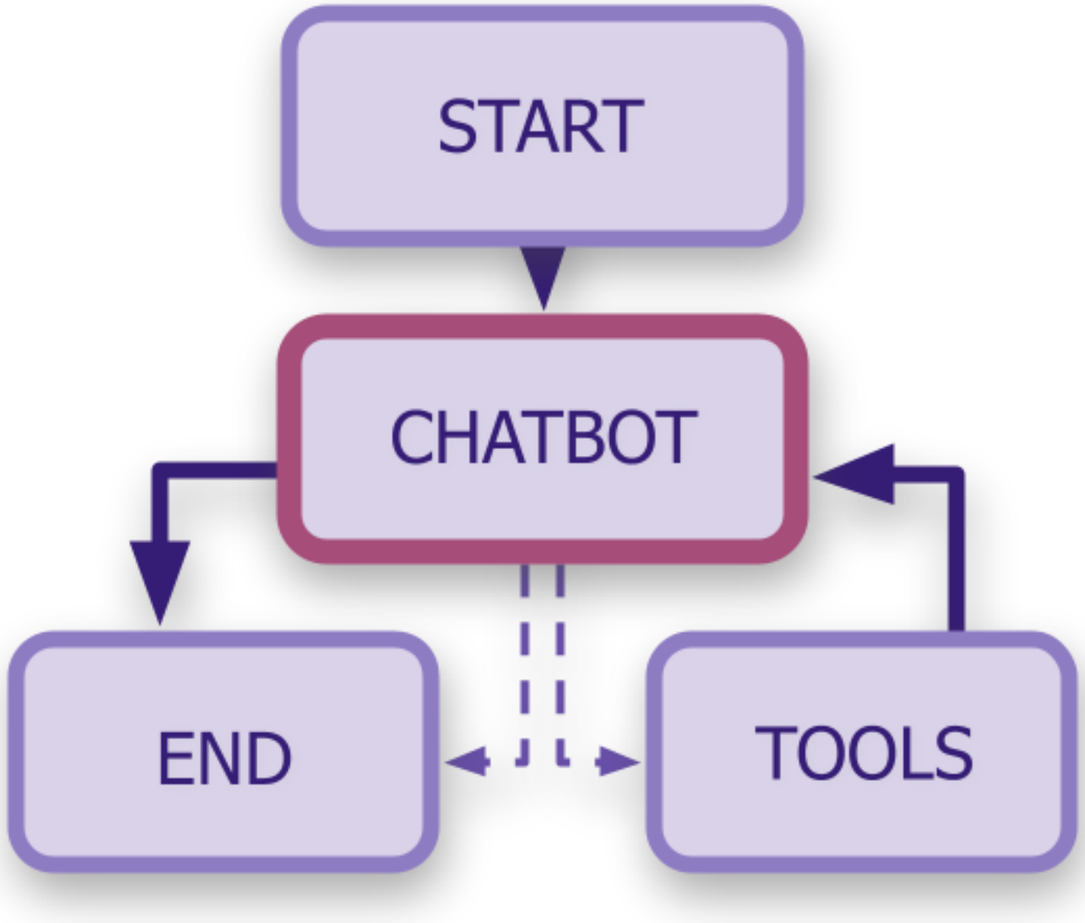
Adding tool nodes

```
# Set up a condition to direct from chatbot
# to tool or end node
graph_builder.add_conditional_edges(
                "chatbot", tools_condition)
```



Adding tool nodes

```
# Set up a condition to direct from chatbot
# to tool or end node
graph_builder.add_conditional_edges(
             "chatbot", tools_condition)


# Connect tools back to chatbot and
# add START and END nodes
graph_builder.add_edge("tools", "chatbot")
graph_builder.add_edge(START, "chatbot")
graph_builder.add_edge("chatbot", END)
```



Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

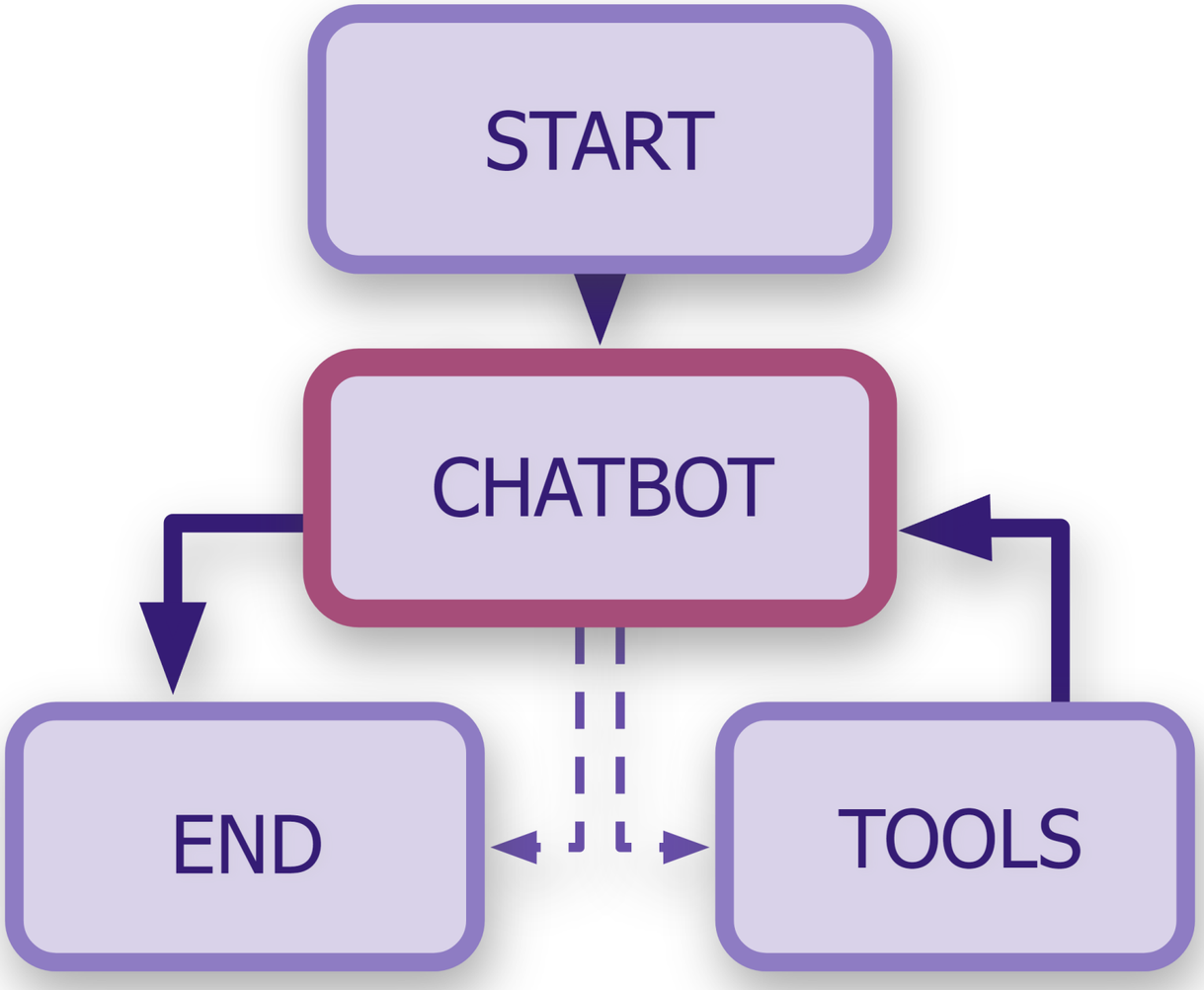
Adding memory and conversation

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

datacamp

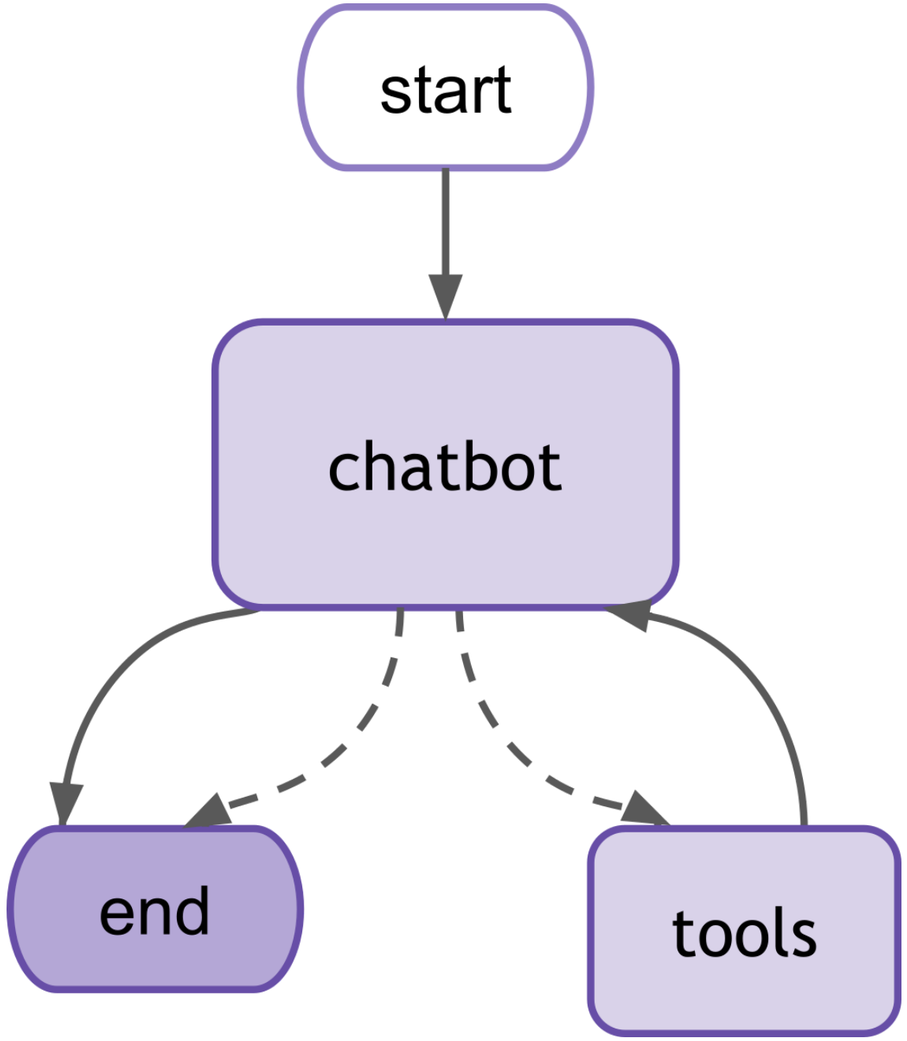
Testing tool use



Testing tool use

```
# Produce the chatbot graph
display(Image(app.get_graph().draw_mermaid_png()))

# Define a function to execute the chatbot, streaming each message
def stream_tool_responses(user_input: str):
    for event in graph.stream({"messages": [("user", user_input)]}):

        # Return the agent's last response
        for value in event.values():
            print("Agent:", value["messages"])

# Define the query and run the chatbot
user_query = "House of Lords"
stream_tool_responses(user_query)
```

Visualizing the diagram



Streaming the output

```
Agent: [AIMessage(content='', additional_kwargs={'tool_calls': [{'function': {'arguments':
'{"query":"House of Lords"}', 'name': 'wikipedia'}, 'type': 'function'}]},
response_metadata={'...'})]
Agent: [ToolMessage(content='Page: House of Lords\nSummary: The House of Lords is the upper
house of the Parliament of the United Kingdom. Like the lower house...The House of Lords
also has a Church of England role...', name='wikipedia', id='...',')]
Agent: [AIMessage(content='The House of Lords is the upper house of the Parliament
of the United Kingdom, located in the Palace of Westminster in London. It is one
of the oldest institutions in the world..., additional_kwargs={},
response_metadata=
{'finish_reason': 'stop', 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint':
'fp_0ba0d124f1'}, id='run-ae3a0b4f-5b42-4f4e-9409-7c191af0b9c9-0')]
```

Adding memory

```
# Import the modules for saving memory
from langgraph.checkpoint.memory import MemorySaver

# Modify the graph with memory checkpointing
memory = MemorySaver()

# Compile the graph passing in memory
graph = graph_builder.compile(checkpointer=memory)
```

Streaming outputs with memory

```
# Set up a streaming function for a single user
def stream_memory_responses(user_input: str):
    config = {"configurable": {"thread_id": "single_session_memory"}}

    # Stream the events in the graph
    for event in graph.stream({"messages": [("user", user_input)]}, config):

        # Return the agent's last response
        for value in event.values():
            if "messages" in value and value["messages"]:
                print("Agent:", value["messages"])

stream_memory_responses("What is the Colosseum?")
stream_memory_responses("Who built it?")
```

Generating output with memory

```
stream_memory_responses("What is the Colosseum?")
```

```
Agent: [AIMessage(content='', additional_kwargs={'tool_calls': [{'index': 0,
'id': '...', 'function': {'arguments': '
{"query":"Colosseum"}', 'name': 'wikipedia'}, ...])]
Agent: [ToolMessage(content='Page: Colosseum\nSummary: The Colosseum is an ancient
amphitheatre in Rome, Italy. It is the largest standing amphitheatre
in the world..)]
Agent: [AIMessage(content='The Colosseum, located in Rome, is the largest ancient
amphitheatre still standing. Built under Emperor Vespasian and completed by his son
Titus, it hosted gladiatorial games and public events. It is also known as the Flavian
Amphitheatre due to its association with the Flavian dynasty.', additional_kwargs={},
response_metadata={'finish_reason': 'stop', 'model_name': 'gpt-4o-mini-...', ...')]
```

Generating output with memory

```
stream_memory_responses("Who built it?")
```

```
Agent: [AIMessage(content='The Colosseum was built by Emperor Vespasian around
72 AD and completed by his successor, Emperor Titus. Later modifications were
made by Emperor Domitian...')]
```

Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN