Analytics Vidhya

Augmented LLMs with Tools

Dipanjan Sarkar

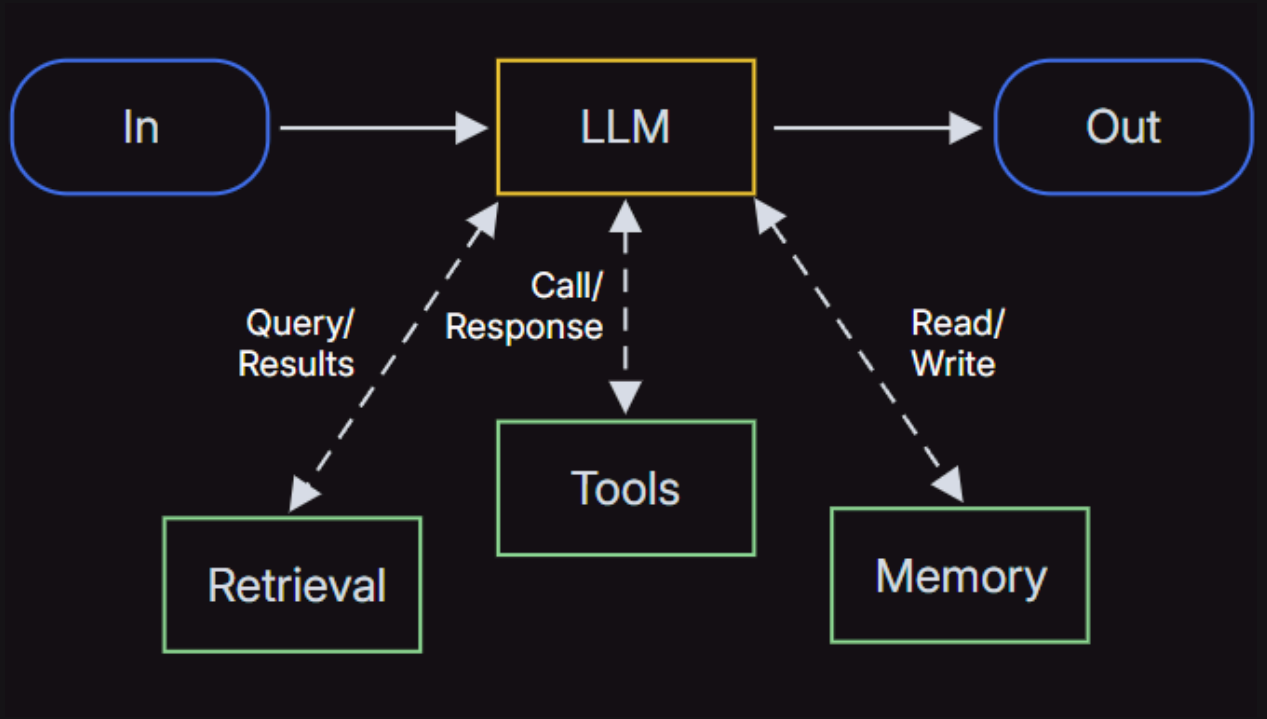
Head of Community & Principal AI Scientist at Analytics Vidhya

Google Developer Expert - ML & Cloud Champion Innovator

Published Author

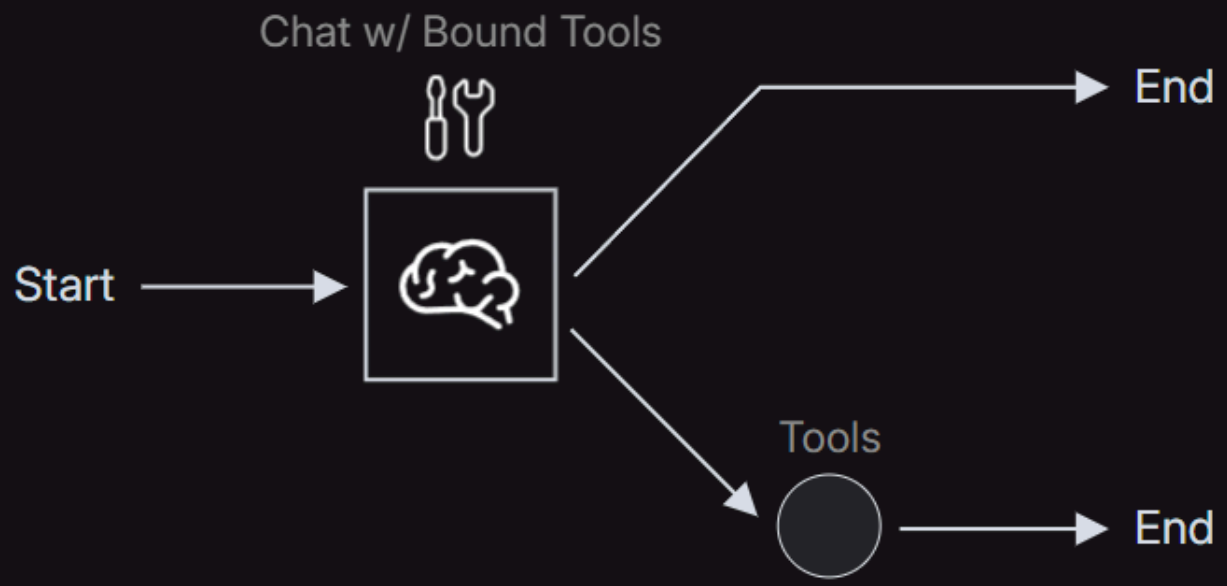
Augmented LLMs - Building Blocks for Agentic Systems

- The basic building block of Agentic AI Systems is an LLM enhanced with augmentations such as retrieval, tools, and memory
- Powerful LLM platforms (like ChatGPT) have these built-in
- When using LLM APIs (like OpenAI GPT-4o) you would need to connect the LLM with relevant tools, memory, and databases
- These augmented LLMs can then generate their search queries, select appropriate tools, and determine what information to retain.



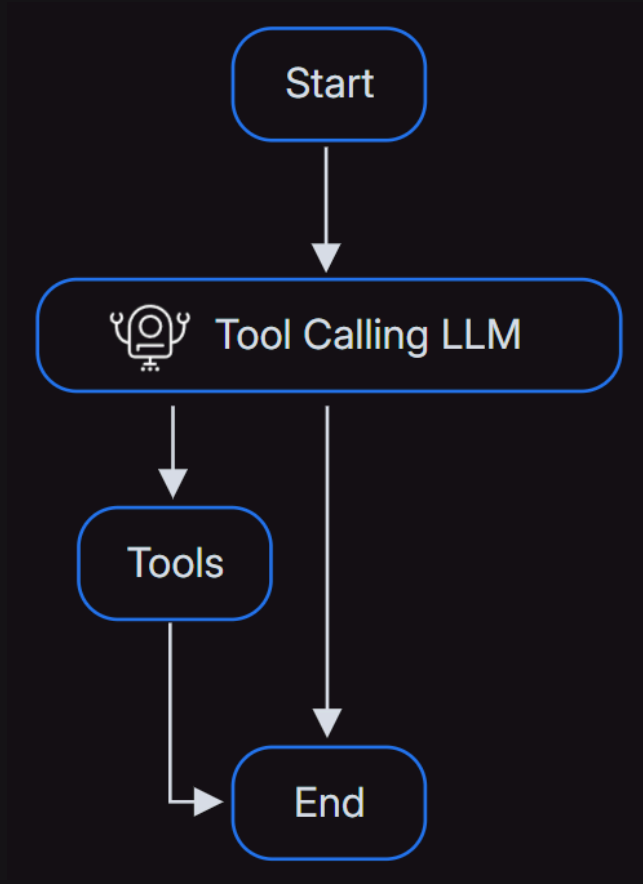
Augmenting LLMs with Tools in LangGraph

- We already know Tools help the LLM interact with external sources of information like web search
- Augmented LLM with Search
  - Here we will build a simple augmented LLM using the capabilities of Tavily Search to allow the LLM to fetch relevant information from the web when necessary.
- Objective
  - Enable the LLM to dynamically decide whether to perform a direct response or use Tavily Search for enriched results in our graph workflow



Augmenting LLMs with Tools in LangGraph - Key Constructs

- **ToolNode**
  - **Purpose**: Acts as a specialized node responsible for executing predefined tools within the graph's workflow.
  - **Functionality**: Upon activation, it processes the current state, identifies any tool calls specified in the messages, and executes the corresponding tools to get the results (ToolMessages)
- **tools_condition**
  - **Purpose**: Serves as a conditional routing function that determines the graph's execution path based on the presence of tool calls after the LLM reasons on the input.
  - **Functionality**: Evaluates the latest message in the state; if it contains tool calls, tools_condition directs the workflow to the ToolNode (Tools) for execution. Otherwise, it routes the workflow to the end node, concluding the process.



Defining Tools and Augmenting LLMs

Key Components

Tavily Search Integration

- Utilize the TavilySearchAPIWrapper to fetch real-time results.
- Configure the search to allow advanced querying, including raw content and specific search depths.

```python
from langchain_openai import ChatOpenAI
from langchain_community.utilities.tavily_search import TavilySearchAPIWrapper
from langchain_core.tools import tool

llm = ChatOpenAI(model="gpt-4o", temperature=0)

tavily_search = TavilySearchAPIWrapper()
@tool
def search_web(query: str, num_results=5):
    """Search the web for a query. Userful for general information or general news"""
    results = tavily_search.raw_results(query=query,
                                        max_results=num_results,
                                        search_depth='advanced',
                                        include_raw_content=True)
    return results

tools = [search_web]
llm_with_tools = llm.bind_tools(tools=tools)

llm_with_tools.invoke('what is the latest news on nvidia')
## OUTPUT
AIMessage(content='', additional_kwargs={'tool_calls': [{'id':
'call_RLYXkGIeEcYOhNT0rEPWgO33', 'function': {'arguments': '{"query":"latest
news on Nvidia","num_results":5}', 'name': 'search_web'}, 'type':
'function'}], 'refusal': None}, ...)
```

Defining Tools and Augmenting LLMs

Key Components

Custom Tool

- Define a search_web tool using LangChain's @tool decorator
- Example: Search for up to 5 results with raw_results()

```python
from langchain_openai import ChatOpenAI
from langchain_community.utilities.tavily_search import TavilySearchAPIWrapper
from langchain_core.tools import tool

llm = ChatOpenAI(model="gpt-4o", temperature=0)

tavily_search = TavilySearchAPIWrapper()
@tool
def search_web(query: str, num_results=5):
    """Search the web for a query. Userful for general information or general news"""
    results = tavily_search.raw_results(query=query,
                                        max_results=num_results,
                                        search_depth='advanced',
                                        include_raw_content=True)
    return results

tools = [search_web]
llm_with_tools = llm.bind_tools(tools=tools)

llm_with_tools.invoke('what is the latest news on nvidia')
## OUTPUT
AIMessage(content='', additional_kwargs={'tool_calls': [{'id':
'call_RLYXkGIeEcYOhNT0rEPWgO33', 'function': {'arguments': '{"query":"latest
news on Nvidia","num_results":5}', 'name': 'search_web'}, 'type':
'function'}], 'refusal': None}, ...)
```

Defining Tools and Augmenting LLMs

Key Components

Binding Tools with LLM

- Combine the tool with a ChatOpenAI model (gpt-4o) using the bind_tools() method and augment the LLM.
- Enables the LLM to decide dynamically when to invoke the tool.

```python
from langchain_openai import ChatOpenAI
from langchain_community.utilities.tavily_search import TavilySearchAPIWrapper
from langchain_core.tools import tool

llm = ChatOpenAI(model="gpt-4o", temperature=0)

tavily_search = TavilySearchAPIWrapper()
@tool
def search_web(query: str, num_results=5):
    """Search the web for a query. Userful for general information or general news"""
    results = tavily_search.raw_results(query=query,
                                        max_results=num_results,
                                        search_depth='advanced',
                                        include_raw_content=True)
    return results

tools = [search_web]
llm_with_tools = llm.bind_tools(tools=tools)

llm_with_tools.invoke('what is the latest news on nvidia')
## OUTPUT
AIMessage(content='', additional_kwargs={'tool_calls': [{'id':
'call_RLYXkGIeEcYOhNT0rEPWgO33', 'function': {'arguments': '{"query":"latest
news on Nvidia","num_results":5}', 'name': 'search_web'}, 'type':
'function'}], 'refusal': None}, ...)
```

Building the Augmented LLM Graph Workflow

Components

State Management

- The State schema includes a messages field with the add_messages reducer to maintain all the messages created across nodes in the graph
- Messages are appended instead of overwritten

```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 langgraph.prebuilt import ToolNode, tools_condition

class State(TypedDict):
    messages: Annotated[list, add_messages]


# Augmented LLM with Tools Node function
def tool_calling_llm(state: State) -> State:
    current_state = state["messages"]
    return {"messages": [llm_with_tools.invoke(current_state)]}

# Build the graph
builder = StateGraph(State)
builder.add_node("tool_calling_llm", tool_calling_llm)
builder.add_node("tools", ToolNode(tools=tools))
builder.add_edge(START, "tool_calling_llm")

# Conditional Edge
builder.add_conditional_edges(
    "tool_calling_llm",
    tools_condition,
    ["tools", END]
)
builder.add_edge("tools", END)
graph = builder.compile()
```

Building the Augmented LLM Graph Workflow

Components

LLM with Tools

- A function node (tool_calling_llm) is defined to process messages using the LLM and invoke tools when needed.

```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 langgraph.prebuilt import ToolNode, tools_condition

class State(TypedDict):
    messages: Annotated[list, add_messages]


# Augmented LLM with Tools Node function
def tool_calling_llm(state: State) -> State:
    current_state = state["messages"]
    return {"messages": [llm_with_tools.invoke(current_state)]}

# Build the graph
builder = StateGraph(State)
builder.add_node("tool_calling_llm", tool_calling_llm)
builder.add_node("tools", ToolNode(tools=tools))
builder.add_edge(START, "tool_calling_llm")

# Conditional Edge
builder.add_conditional_edges(
    "tool_calling_llm",
    tools_condition,
    ["tools", END]
)
builder.add_edge("tools", END)
graph = builder.compile()
```

Building the Augmented LLM Graph Workflow

Components

ToolNode

- A prebuilt node specifically designed for executing tools like search, computation, or external integrations. (search in this case)

```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 langgraph.prebuilt import ToolNode, tools_condition

class State(TypedDict):
    messages: Annotated[list, add_messages]


# Augmented LLM with Tools Node function
def tool_calling_llm(state: State) -> State:
    current_state = state["messages"]
    return {"messages": [llm_with_tools.invoke(current_state)]}

# Build the graph
builder = StateGraph(State)
builder.add_node("tool_calling_llm", tool_calling_llm)
builder.add_node("tools", ToolNode(tools=tools))
builder.add_edge(START, "tool_calling_llm")

# Conditional Edge
builder.add_conditional_edges(
    "tool_calling_llm",
    tools_condition,
    ["tools", END]
)
builder.add_edge("tools", END)
graph = builder.compile()
```

Building the Augmented LLM Graph Workflow

Components

Conditional Edge

- Uses tools_condition to route the flow
  - If the latest LLM response indicates a tool call → Route to the tools node.
  - Otherwise, end the workflow.

```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 langgraph.prebuilt import ToolNode, tools_condition

class State(TypedDict):
    messages: Annotated[list, add_messages]


# Augmented LLM with Tools Node function
def tool_calling_llm(state: State) -> State:
    current_state = state["messages"]
    return {"messages": [llm_with_tools.invoke(current_state)]}

# Build the graph
builder = StateGraph(State)
builder.add_node("tool_calling_llm", tool_calling_llm)
builder.add_node("tools", ToolNode(tools=tools))
builder.add_edge(START, "tool_calling_llm")

# Conditional Edge
builder.add_conditional_edges(
    "tool_calling_llm",
    tools_condition,
    ["tools", END]
)
builder.add_edge("tools", END)
graph = builder.compile()
```

Executing the Augmented LLM Graph Workflow

Two Scenarios Possible

- Non-Tool Call Flow
  - The graph handles a straightforward user query.
  - LLM generates a direct response without invoking external tools.
- Tool Call Flow
  - The graph identifies that the user query requires an external tool (e.g., search_web)
  - The appropriate tool is invoked, and its results are returned

```
## Non Tool Call Flow:
user_input = "Explain AI in 2 bullets"
for event in graph.stream({"messages": user_input},
                          stream_mode='values'):
    event['messages'][-1].pretty_print()
## OUTPUT:
========== Human Message ==========

Explain AI in 2 bullets
========== Ai Message ==========

- **Definition and Functionality**: Artificial Intelligence (AI) refers to the
simulation of human intelligence ...

- **Applications and Impact**: AI is utilized in various applications such as
natural language processing, robotics, and computer vision,...


## Tool Call Flow:
user_input = "What is the latest news on OpenAI product releases"
for event in graph.stream({"messages": user_input},
                          stream_mode='values'):
    event['messages'][-1].pretty_print()
## OUTPUT:
============ Human Message =============

What is the latest news on OpenAI product releases
============ Ai Message ===============
Tool Calls:
  search_web (call_hvznrIeGAS2Qh9ohprVbkMoj)
 Call ID: call_hvznrIeGAS2Qh9ohprVbkMoj
  Args:
    query: OpenAI latest product releases
    num_results: 5
============ Tool Message =============
Name: search_web

{"query": "OpenAI latest product releases", "follow_up_questions": null, "answer":
null, "images": [], "results": [{"title": "OpenAI teases 12 days of mystery product
launches starting tomorrow", "url": "https://arstechnica.com/ai/2024/12/openai-
teases-12-days-of-mystery-product-launches-starting-tomorrow/", "content": "OpenAI
teases 12 days of mystery product launches starting tomorrow ..."}]}
```

Executing the Augmented LLM Graph Workflow

We don't send these results back to the LLM, though; this loop creates an agentic flow.

We will address this in the next video...

```
## Non Tool Call Flow:
user_input = "Explain AI in 2 bullets"
for event in graph.stream({"messages": user_input},
                          stream_mode='values'):
    event['messages'][-1].pretty_print()
## OUTPUT:
========== Human Message ==========

Explain AI in 2 bullets
========== Ai Message ===========

- **Definition and Functionality**: Artificial Intelligence (AI) refers to the
simulation of human intelligence ...

- **Applications and Impact**: AI is utilized in various applications such as
natural language processing, robotics, and computer vision,...


## Tool Call Flow:
user_input = "What is the latest news on OpenAI product releases"
for event in graph.stream({"messages": user_input},
                          stream_mode='values'):
    event['messages'][-1].pretty_print()
## OUTPUT:
============ Human Message =============

What is the latest news on OpenAI product releases
============ Ai Message ===============
Tool Calls:
  search_web (call_hvznrIeGAS2Qh9ohprVbkMoj)
 Call ID: call_hvznrIeGAS2Qh9ohprVbkMoj
  Args:
    query: OpenAI latest product releases
    num_results: 5
============ Tool Message =============
Name: search_web

{"query": "OpenAI latest product releases", "follow_up_questions": null, "answer":
null, "images": [], "results": [{"title": "OpenAI teases 12 days of mystery product
launches starting tomorrow", "url": "https://arstechnica.com/ai/2024/12/openai-
teases-12-days-of-mystery-product-launches-starting-tomorrow/", "content": "OpenAI
teases 12 days of mystery product launches starting tomorrow ..."}]}
```

Thanks