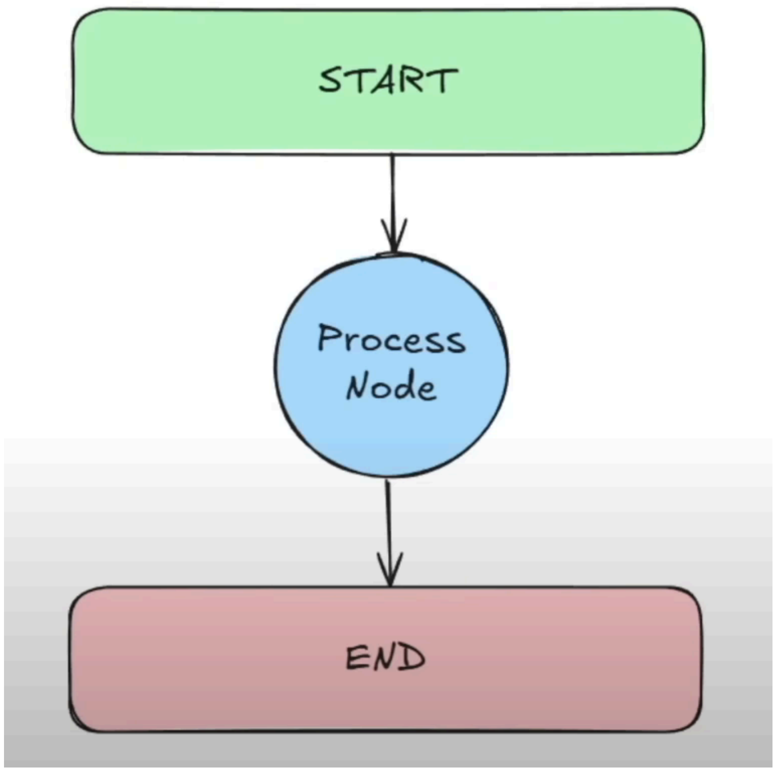
AI Chatbot Using LangGraph

Simple LLM Integration

Goal: Integrate LLMs in Graphs

Objectives:

1. Define state structure with a list of HumanMessage objects.
2. Initialize a GPT-4o model using LangChain's ChatOpenAI
3. Sending and handling different types of messages
4. Building and compiling the graph of the Agent



Code:

```
from langchain_groq import ChatGroq
from typing import TypedDict, List
from langchain_core.messages import HumanMessage
from langgraph.graph import StateGraph, START, END
from dotenv import load_dotenv

load_dotenv()
```

```
class AgentState(TypedDict):
    messages : List[HumanMessage]
```

- Tells LangGraph that these are human messages

```
llm = ChatGroq(model="openai/gpt-oss-120b")
```

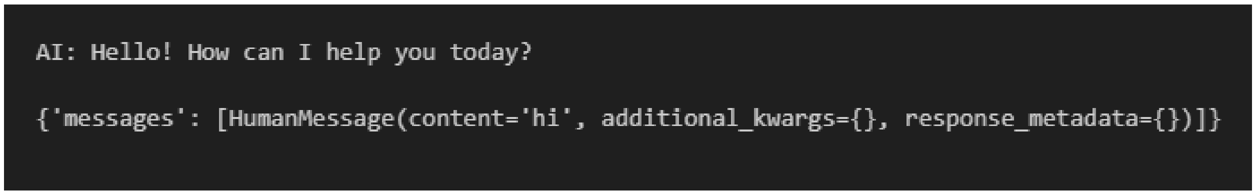
```
def process(state:AgentState) → AgentState:
    response = llm.invoke(state["messages"])
    print (f"\nAI: {response.content}")
    return state
```

```
graph = StateGraph(AgentState)

graph.add_node("process", process)

graph.add_edge(START, "process")

graph.add_edge("process", END)
```

```
agent= graph.compile()
```

Invoke:

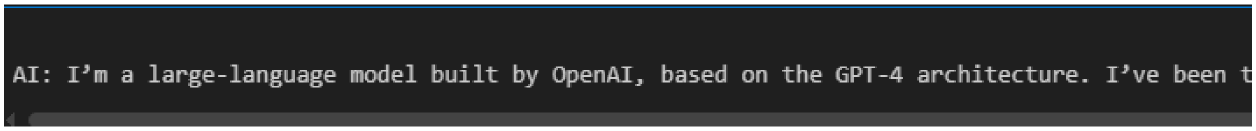
```
user_input = input("Enter: ")

agent.invoke({"messages": [HumanMessage(content=user_input)]})
```



```
AI: Hello! How can I help you today?

{'messages': [HumanMessage(content='hi', additional_kwargs={}, response_metadata={})]}
```

Alt method:

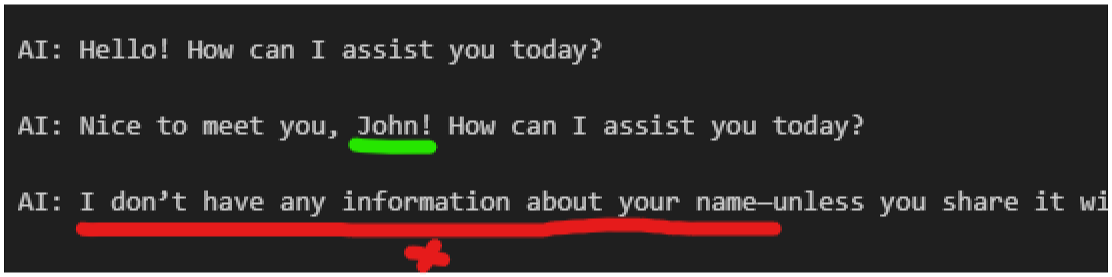
```
user_input = "Which llm model are you?"

agent.invoke({"messages": [HumanMessage(content=user_input)]})
```



```
AI: I’m a large-language model built by OpenAI, based on the GPT-4 architecture. I’ve been t
```

Loop (No History):

```
user_input = input("Human: ")

while user_input != "exit":
    agent.invoke({"messages": [HumanMessage(content=user_input)]})
    user_input = input("Human ")
```



```
AI: Hello! How can I assist you today?

AI: Nice to meet you, John! How can I assist you today?

AI: I don't have any information about your name—unless you share it wi
```

- If you input `"exit"`, the loop will stop

Chatbot with History

```
from langchain_groq import ChatGroq
from typing import TypedDict, List, Union
from langchain_core.messages import HumanMessage, AIMessage
from langgraph.graph import StateGraph, START, END
from dotenv import load_dotenv

load_dotenv()
```

`Union` :

- `Union[X, Y]` means a value **can be either type** `X` **or type** `Y`.
- `Union[int, float, str]` → accepts either an `int`, `float`, or `str`.

🆕 Python 3.10+: New Syntax

Starting in **Python 3.10**, you can write unions using the pipe `|` operator:

```
def stringify(data: int | float | str) → str:
    return str(data)
```

```
class AgentState(TypedDict):
    messages: List[Union[HumanMessage, AIMessage]]
```

- The value for `messages` must be a list (`List[...]`) that contains elements of **either** `HumanMessage` **or** `AIMessage`.

> The `AgentState` dictionary must have a key called `"messages"` whose value is a **list of messages**, where each message is either a `HumanMessage` or an `AIMessage`.

```
llm = ChatGroq(model="openai/gpt-oss-120b")
```

```
def process(state: AgentState) → AgentState:
    """This node will solve the request you input"""

    response = llm.invoke(state["messages"])
    state["messages"].append(AIMessage(content=response.content))
    print(f"AI: {response.content}")
    return state
```

`state["messages"].append(AIMessage(content=response.content))`:

- **`response.content` → Only response text. No metadata**
- We are appending the newly generated AI response to **`state["messages"]` under `AIMessage`**

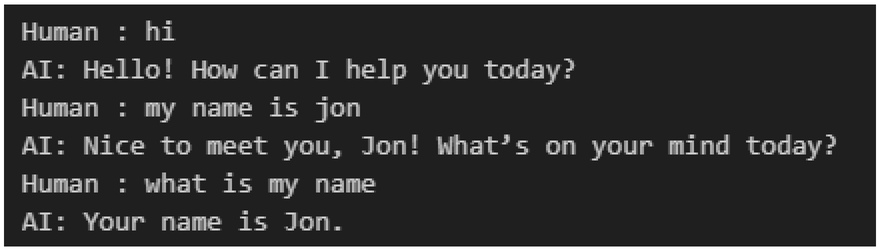
💡 **This receives the entire message history(*we'll provide later*) & generates response and adds it to `messages`**

```
graph = StateGraph(AgentState)

graph.add_node("process", process)

graph.add_edge(START, "process")

graph.add_edge("process", END)

agent= graph.compile()
```

- Same graph as previous one

```
conversation_history = []

user_input = input("Human: ")

while user_input != "exit":

    conversation_history.append(HumanMessage(content=user_input))

    print(f"Human : {user_input}")

    result = agent.invoke({"messages": conversation_history}) #Provides the en
tire history


    conversation_history = result["messages"] #conversation_history is being r
```

```
eplaced


    user_input = input("Human: ")
```



```
Human : hi
AI: Hello! How can I help you today?
Human : my name is jon
AI: Nice to meet you, Jon! What’s on your mind today?
Human : what is my name
AI: Your name is Jon.
```

- `conversation_history = []` → **This is the memory**
- In `process` node, we had appended the `AIMessage` , here we append `HumanMessage`
- So, The Flow:
    1. Get Human message through `user_input`
    2. Append it in `conversation_history`
    3. Print `HuamnMessage`
    4. **INVOKE THE AGENT**
        - This will execute the `process` **node**
        - It appends and print the `AIMessage`

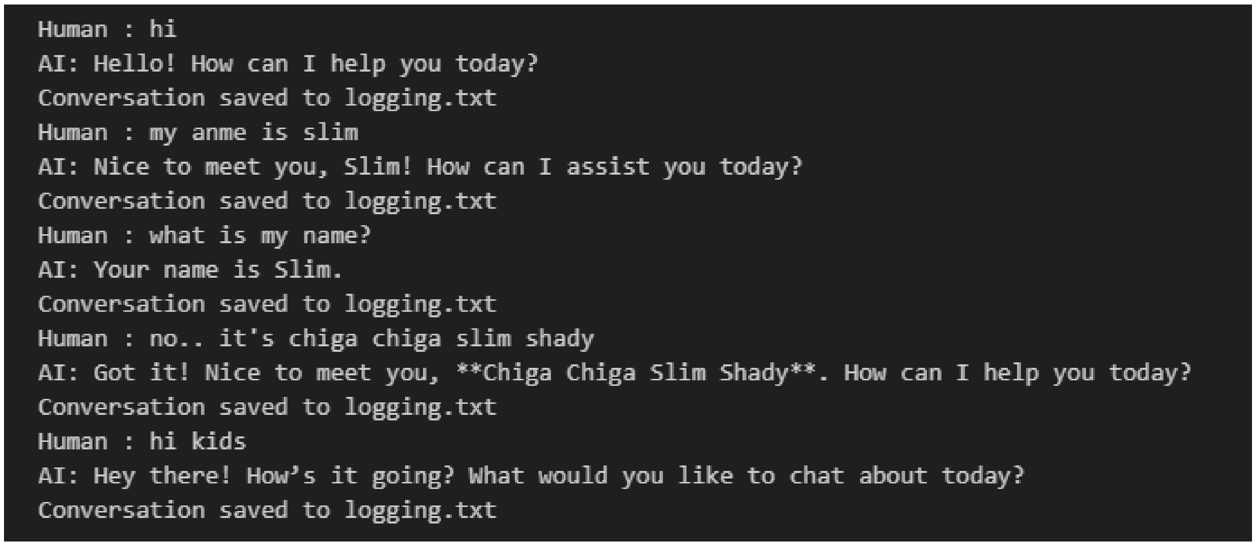
📊 Visual Flow

```
[USER TYPES MESSAGE]
       ↓
+----------------------+
| conversation_history|
+----------------------+
       ↓
```

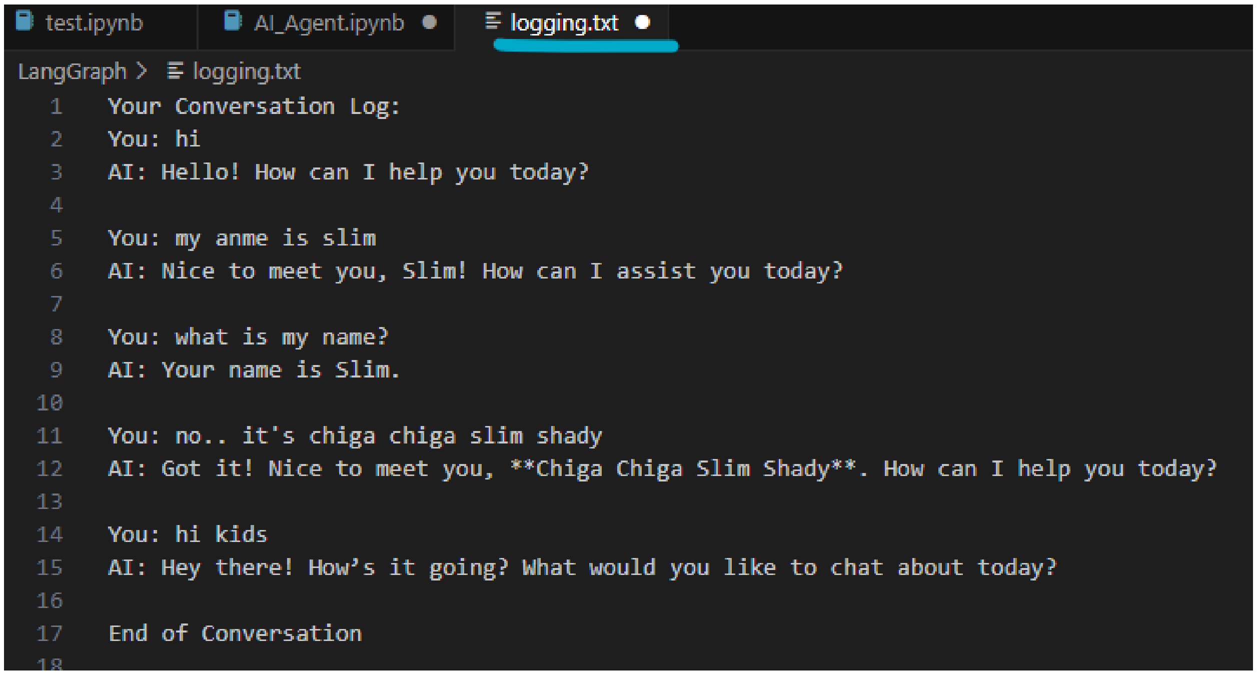
```
agent.invoke({"messages": history})
      ↓
LANGGRAPH starts:
  START → process → END
      ↓
process(state):
  - Calls LLM: llm.invoke(messages)
  - Adds AI reply to history
  - Returns updated state
      ↓
agent returns result
      ↓
Update conversation_history with new messages
      ↓
Loop again (user input)...
```

Save the conversation in txt file

```
with open("logging.txt", "w") as file:
    file.write("Your Conversation Log:\n")

    for message in conversation_history:
        if isinstance(message, HumanMessage):
            file.write(f"You: {message.content}\n")

        elif isinstance(message, AIMessage):
            file.write(f"AI: {message.content}\n\n")
    file.write("End of Conversation")

print("Conversation saved to logging.txt")
```



```
Human : hi
AI: Hello! How can I help you today?
Conversation saved to logging.txt
Human : my anme is slim
AI: Nice to meet you, Slim! How can I assist you today?
Conversation saved to logging.txt
Human : what is my name?
AI: Your name is Slim.
Conversation saved to logging.txt
Human : no.. it's chiga chiga slim shady
AI: Got it! Nice to meet you, **Chiga Chiga Slim Shady**. How can I help you today?
Conversation saved to logging.txt
Human : hi kids
AI: Hey there! How’s it going? What would you like to chat about today?
Conversation saved to logging.txt
```



```
Your Conversation Log:
You: hi
AI: Hello! How can I help you today?

You: my anme is slim
AI: Nice to meet you, Slim! How can I assist you today?

You: what is my name?
AI: Your name is Slim.

You: no.. it's chiga chiga slim shady
AI: Got it! Nice to meet you, **Chiga Chiga Slim Shady**. How can I help you today?

You: hi kids
AI: Hey there! How’s it going? What would you like to chat about today?

End of Conversation
```