# **Project Report: Intelligent Chatbot Using LangGraph, Groq LLM, Tavily Search, and Streamlit**

## **1. Introduction**

This project is an implementation of an intelligent chatbot system that combines multiple powerful technologies to answer user queries in real-time. The chatbot uses **LangGraph** for managing conversation flow, **Groq's LLM** (specifically openai/gpt-oss-20b) to generate responses, **Tavily Search API** to fetch live data from the internet, and **Streamlit** to build an interactive front-end interface.

The goal of the project is to create a modular, extendable, and real-time chatbot that not only handles simple questions but can also fetch live data when necessary.

## **2. Tech Stack**

The following technologies are used in this project:

* **LangGraph**: Manages the state transitions between the agent and tools.
* **Groq LLM**: An open-source large language model (GPT OSS 20B) used to generate responses.
* **Tavily Search API**: Tool for live search queries, enabling real-time data retrieval.
* **Streamlit**: A Python-based framework used to build an intuitive user interface for interacting with the chatbot.

## **3. Architecture Overview**

Below is a high-level architecture and workflow of the chatbot:

1. **User Input**: The user enters a question via the Streamlit interface.
2. **LLM Agent**: The message is passed to Groq’s LLM, which analyzes the input.
3. **Decision Layer**:  
   * If the model can answer directly, it generates a response.
   * If it determines more information is needed, it calls the **Tavily Search tool**.
4. **Tool Node (Tavily)**: Fetches real-time web search results.
5. **LLM Response**: Uses the results to formulate an answer.
6. **Response Rendering**: The final response is shown to the user in Streamlit.

## **4. Workflow and Logic Implementation**

The logic and flow are implemented using **LangGraph's StateGraph** API.

### **Class Definition: chatbot**

The chatbot class encapsulates the entire functionality of the bot.

### **Key Components:**

**Initialization**:  
  
 self.llm = ChatGroq(model\_name="openai/gpt-oss-20b")

**Tool Binding**:  
 Tavily Search is initialized and bound to the agent.  
  
 tool = TavilySearchResults(max\_results=2)

self.llm\_with\_tool = self.llm.bind\_tools([tool])

**Agent Node**:  
 Handles message invocation with the LLM.  
  
 def call\_model(self, state: MessagesState):

response = self.llm\_with\_tool.invoke(state['messages'])

**Router Function**:  
 Determines whether to call a tool or end the conversation.  
  
 def router\_function(self, state: MessagesState) -> Literal["tools", END]:

if last\_message.tool\_calls:

return "tools"

return END

**LangGraph Workflow**:  
  
 workflow = StateGraph(MessagesState)

workflow.add\_node("agent", self.call\_model)

workflow.add\_node("tools", self.tool\_node)

workflow.add\_edge(START, "agent")

workflow.add\_conditional\_edges("agent", self.router\_function, {"tools": "tools", END: END})

workflow.add\_edge("tools", "agent")

## **5. Streamlit Interface**

The chatbot is made interactive using Streamlit. The app consists of:

* A title and short description.
* A text input for the user's question.
* A button to trigger the chatbot.
* A display area for the chatbot's response.

### **Sample Streamlit Code:**

st.title("ChatBot with LangGraph")

question = st.text\_input("Enter your question here:")

if st.button("Get Answer"):

response = workflow.invoke({"messages": [question]})

st.write("\*\*Answer:\*\*", response['messages'][-1].content)

## **6. Example Queries**

### **✅ Basic Knowledge:**

**Q**: "What is the capital of France?"  
 **A**: "The capital of France is Paris."

### **🌐 Real-Time Query:**

**Q**: "What is the latest news about SpaceX?"  
 **A**: (Uses Tavily tool to search the latest news and summarizes it.)

### **🐱 Fun Query:**

**Q**: "Tell me a fun fact about cats."  
 **A**: "Cats can rotate their ears 180 degrees to pinpoint sounds."

## **7. Future Scope**

The chatbot architecture is modular and can be extended in multiple ways:

* **Add Memory Support**: Allow the chatbot to remember past conversation context.
* **Integrate More Tools**: For example, a weather tool, translation, or calculator.
* **Deploy as a Web App**: Host it using platforms like Streamlit Cloud, Vercel, or HuggingFace Spaces.
* **Multilingual Support**: Enable responses in different languages.

## **8. Conclusion**

This project successfully demonstrates how **LangGraph**, **Groq LLM**, **Tavily Search**, and **Streamlit** can be combined to build a functional, real-time, intelligent chatbot. The framework is lightweight, customizable, and capable of handling both static and dynamic queries with ease.

## **9. References**

* LangGraph Documentation
* Groq LLM
* Tavily API
* Streamlit