Building a Patient Information Collection and Advisory Chatbot Agent

1. Objectives

- Develop an AI-powered chatbot agent that interactively collects patient information such as name, age, and symptoms.
- Provide preliminary health advice based on the collected patient data.
- Utilize **LangGraph** to manage the conversational flow and state.
- Employ **AzureChatOpenAI** as the language model through **LangChain**.
- Optionally integrate **Tavily** to fetch real-time information to supplement chatbot responses.

2. Problem Statement

Patients often need quick, preliminary health assessments before seeing healthcare professionals. This exercise aims to build an AI-driven chatbot agent that:

- Engages users in a natural conversational manner to gather essential patient details.
- Analyzes the collected information to offer relevant, preliminary health advice or recommendations.
- Optionally enhances advice with real-time web information via Tavily.
- Improves user experience with interactive, guided dialogue flow controlled by LangGraph.

3. Inputs / Shared Artifacts

- Python 3.x environment (preferably create virtual enviroment).
- Installed packages: langchain, langgraph, openai, and optionally tavily.
- Azure OpenAI API access with deployment configured.
- (Optional) Tavily API key for real-time web search integration.
- Basic understanding of prompt engineering and conversational AI design.

4. Expected Outcomes

- A conversational chatbot agent that collects patient details interactively.
- The agent provides preliminary health advice based on user input.
- When integrated, it supplements advice with relevant real-time information.
- A documented Jupyter Notebook showcasing the full implementation and sample interactions.
- Hands-on experience with LangGraph for managing complex conversation flows.
- Practical skills using AzureChatOpenAI for conversational AI.

5. Concepts Covered

- **Conversational AI:** Building a chatbot that interacts naturally with users.
- **Conversation Flow Management:** Using LangGraph to design and control dialogue states.
- **Language Model Integration:** Leveraging AzureChatOpenAI through LangChain for generating context-aware responses.
- **Optional Real-Time Data Retrieval:** Using Tavily to provide dynamic, up-to-date information during the conversation.
- **Prompt Engineering:** Crafting effective prompts for health-related advice generation.

6. Example: Step-by-Step Instructions and Code Demo

```
# !pip install langchain-openai langchain-community langchain_openai
langchain-tavily langgraph faiss-cpu
from langchain.docstore.document import Document
from langgraph.graph import StateGraph, MessagesState, START, END
from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_openai import AzureChatOpenAI, AzureOpenAIEmbeddings
from langchain.vectorstores import FAISS
from langgraph.prebuilt import ToolNode
from langchain_core.messages import SystemMessage, HumanMessage


mock_chunks = [
    Document(
        page_content="Patients with a sore throat should drink warm fluids
and avoid cold beverages."
    ),
    Document(
        page_content="Mild fevers under 38.5°C can often be managed with rest
and hydration."
    ),
    Document(
        page_content="If a patient reports dizziness, advise checking their
blood pressure and hydration level."
    ),
    Document(
        page_content="Persistent coughs lasting more than 2 weeks should be
evaluated for infections or allergies."
    ),
    Document(
        page_content="Patients experiencing fatigue should consider iron
deficiency or poor sleep as potential causes."
    ),
]

os.environ["AZURE_OPENAI_EMBEDDING_API_KEY"] = ""
os.environ["AZURE_OPENAI_EMBEDDING_ENDPOINT"] = "https://aiportalapi.stu-platform.live/jpe"
os.environ["AZURE_OPENAI_EMBED_MODEL"] = "text-embedding-3-small"

os.environ["AZURE_OPENAI_LLM_API_KEY"] = ""
os.environ["AZURE_OPENAI_LLM_ENDPOINT"] = "https://aiportalapi.stu-platform.live/jpe"
os.environ["AZURE_OPENAI_LLM_MODEL"] = "GPT-4o-mini"

os.environ["TAVILY_API_KEY"] = ""
# --- Setup LLM ---
llm = AzureChatOpenAI(
    azure_endpoint=os.environ["AZURE_OPENAI_ENDPOINT"],
    api_key=os.getenv("AZURE_OPENAI_API_KEY"),
    azure_deployment=os.getenv("AZURE_DEPLOYMENT_NAME"),
    api_version="2024-02-15-preview",
)

# --- Setup FAISS Retriever from Mock Chunks ---
embedding_model = AzureOpenAIEmbeddings(
    model=os.getenv("AZURE_OPENAI_EMBED_MODEL"),
    api_key=os.getenv("AZURE_OPENAI_EMBEDDING_API_KEY"),
    azure_endpoint=os.getenv("AZURE_OPENAI_EMBEDDING_ENDPOINT"),
    api_version="2024-02-15-preview",
)
db = FAISS.from_documents(mock_chunks, embedding_model)
retriever = db.as_retriever()

# --- TOOL 1: RETRIEVE ADVICE ---
@tool
def retrieve_advice(user_input: str) -> str:
    """Searches internal documents for relevant patient advice."""
    docs = retriever.get_relevant_documents(user_input)
    return "\n".join(doc.page_content for doc in docs)

# --- TOOL 2: TAVILY SEARCH ---
tavily_tool = TavilySearchResults()
```

```
# --- LLM SETUP ---
llm = AzureChatOpenAI(
    azure_endpoint=os.environ["AZURE_OPENAI_LLM_ENDPOINT"],
    api_key=os.getenv("AZURE_OPENAI_LLM_API_KEY"),
    azure_deployment=os.getenv("AZURE_OPENAI_LLM_MODEL"),
    api_version="2024-02-15-preview",
)

llm_with_tools = llm.bind_tools([retrieve_advice, tavily_tool])

# --- MODEL NODE ---
def call_model(state: MessagesState):
    messages = state["messages"]
    response = llm_with_tools.invoke(messages)
    return {"messages": [response]}

# --- CONDITIONAL ROUTING ---
def should_continue(state: MessagesState):
    last_message = state["messages"][-1]
    if last_message.tool_calls:
        return "tools"
    return END

# --- TOOL NODE ---
tool_node = ToolNode([retrieve_advice, tavily_tool])

# --- BUILD THE GRAPH ---
graph_builder = StateGraph(MessagesState)
graph_builder.add_node("call_model", call_model)
graph_builder.add_node("tools", tool_node)

graph_builder.add_edge(START, "call_model")
graph_builder.add_conditional_edges("call_model", should_continue, ["tools",
END])
graph_builder.add_edge("tools", "call_model")
```

```
graph = graph_builder.compile()
if __name__ == "__main__":
    result = graph.invoke(
        {
            "messages": [
                SystemMessage(
                    content="You are a helpful medical assistant. Use tools
if needed."
                ),
                HumanMessage(
                    content="I feel tired and have a sore throat. What should
I do?"
                ),
            ]
        }
    )

    print("Final Response:")
    print(result["messages"][-1].content)
```

7. Final Submission Checklist

- Submit your Python script containing the full code in .py file.
- Use a dummy input list (auto input). Do not use manual input (input() built-in).
- Include sample inputs and outputs.