Code:-

```
import os
from dotenv import load_dotenv
from langgraph.graph import StateGraph, START, END
from typing import TypedDict, Annotated
from langchain_core.messages import HumanMessage, AIMessage
from langgraph.graph.message import add_messages
from langgraph.checkpoint.memory import MemorySaver
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import spacy
from transformers import pipeline
# Load environment variables
load_dotenv()
# NLP pipelines and memory saver initialization
nlu_pipeline = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
nlp = spacy.load("en_core_web_sm")
sentiment_pipeline = pipeline("sentiment-analysis")
memory = MemorySaver()
# Contextual memory class to store user-specific data
class ContextualMemory(MemorySaver):
    def __init__(self):
        super().__init__()
        self.user_context = {}
    def store_user_context(self, user_id, context):
        self.user_context[user_id] = context
    def get_user_context(self, user_id):
        return self.user_context.get(user_id, {})
contextual_memory = ContextualMemory()
# User profile management class
class UserProfile:
    def __init__(self):
        self.profiles = {}
    def update_profile(self, user_id, preference):
        if user_id not in self.profiles:
            self.profiles[user_id] = {}
        self.profiles[user_id].update(preference)
    def get_profile(self, user_id):
        return self.profiles.get(user_id, {})
user_profiles = UserProfile()
# Feedback collector class
class FeedbackCollector:
    def __init__(self):
        self.feedback = []
    def collect_feedback(self, feedback):
        self.feedback.append(feedback)
```

```
feedback_collector = FeedbackCollector()
# Define State class
class State(TypedDict):
    messages: Annotated[list, add_messages]
# Initialize the graph builder
graph_builder = StateGraph(State)
# Variables to store the trained model and its accuracy
trained_model = None
model_accuracy = None
# Create separate nodes for intent recognition, sentiment analysis, LLM, etc.
def recognize_intent_node(state: State):
    user_input = state["messages"][-1].content
    candidate_labels = ["greeting", "request", "question", "feedback", "fetch", "train", "predict",
"load dataset"]
    intent = nlu_pipeline(user_input, candidate_labels)
    recognized_intent = intent["labels"][0] if intent["scores"][0] > 0.5 else "unknown"
    state["messages"].append(AIMessage(content=f"Recognized Intent: {recognized_intent}"))
    return state
def extract_entities_node(state: State):
    user_input = state["messages"][-1].content
    doc = nlp(user_input)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    entity_text = ", ".join([f"{ent[0]} ({ent[1]})" for ent in entities])
    state["messages"].append(AIMessage(content=f"Extracted Entities: {entity_text}"))
    return state
def analyze_sentiment_node(state: State):
    user_input = state["messages"][-1].content
    sentiment_result = sentiment_pipeline(user_input)[0]
    sentiment_analysis = {"label": sentiment_result["label"], "score": sentiment_result["score"]}
    state["messages"].append(AIMessage(content=f"Sentiment: {sentiment_analysis['label']} with
score {sentiment_analysis['score']}"))
    return state
# Node to load the dataset and train the model
def train_model_node(state: State):
    global trained_model, model_accuracy
    user_input = state["messages"][-1].content

    # Check if intent is "load dataset"
    if "load dataset" in user_input:
        dataset_path = "/Users/haresh.sam/Downloads/CLEANING_TASK.xlsx"  # Use the correct
path
        data = pd.read_excel(dataset_path)  # Load Excel dataset

        # Assuming the dataset has a 'target' column for classification
        X = data.drop(columns=["target"])
        y = data["target"]

        # Split the dataset into train and test sets (80% train, 20% test)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        # Initialize and train the model (Random Forest as an example)
        model = RandomForestClassifier(n_estimators=100)
        model.fit(X_train, y_train)

        # Test the model on the test set
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)

        trained_model, model_accuracy = model, accuracy
        state["messages"].append(AIMessage(content=f"Model trained with an accuracy of
{accuracy * 100:.2f}%"))
    else:
        state["messages"].append(AIMessage(content="Please specify 'load dataset' to train the
model."))

    return state
# LLM node to generate dynamic responses using the language model
def llm_node(state: State):
    state["messages"].append(AIMessage(content="LLM is not enabled in this version."))
    return state
def chatbot_logic_node(state: State):
    user_input = state["messages"][-1].content
    # Handle feedback collection
    if "Was this helpful?" in user_input:
        feedback_collector.collect_feedback(user_input)
        state["messages"].append(AIMessage(content="Thank you for your feedback!"))
    else:
        # Ask for feedback after response
        state["messages"].append(AIMessage(content="I hope that was helpful. Was this response
useful?"))
    return state
# Build graph by adding all custom nodes and defining edges
graph_builder.add_node("intent_recognition", recognize_intent_node)
graph_builder.add_node("entity_extraction", extract_entities_node)
graph_builder.add_node("sentiment_analysis", analyze_sentiment_node)
graph_builder.add_node("train_model", train_model_node)
graph_builder.add_node("llm", llm_node)
graph_builder.add_node("chatbot_logic", chatbot_logic_node)
# Define the flow of the nodes
graph_builder.add_edge(START, "intent_recognition")
graph_builder.add_edge("intent_recognition", "entity_extraction")
graph_builder.add_edge("entity_extraction", "sentiment_analysis")
graph_builder.add_edge("sentiment_analysis", "train_model")
graph_builder.add_edge("train_model", "llm")
graph_builder.add_edge("llm", "chatbot_logic")
graph_builder.add_edge("chatbot_logic", END)
# Compile the graph
graph = graph_builder.compile(checkpointer=contextual_memory)
# Run the chatbot in a loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break
    config = {"configurable": {"thread_id": "1"}}
    events = graph.stream({"messages": [HumanMessage(content=user_input)]}, config,
stream_mode="values")
    for event in events:
        print(event["messages"][-1].content)
```

LangGraph Documentation

1. Overview

This script demonstrates a chatbot framework using **LangGraph** to build a **StateGraph**, integrating various natural language processing (NLP) tasks like intent recognition, entity extraction, sentiment analysis, and machine learning (ML) model training. The chatbot processes user input through these stages and interacts with a memory system to store user context, enhancing future interactions.

2. Key Components and Libraries

- **LangGraph**: A library used to build dynamic state machines (graphs) that handle various tasks in sequence, ensuring smooth transitions between tasks.
- **Spacy & Transformers**: NLP tools used for entity extraction, sentiment analysis, and intent recognition.
- **LangChain**: Interfacing LLMs (Large Language Models) for dynamic response generation (though not fully enabled in this code).
- **MemorySaver**: A checkpointing system that stores contextual information, allowing the chatbot to remember previous conversations.
- **RandomForestClassifier**: A machine learning model used for training and testing a dataset for classification tasks.

3. Flow of Execution (Algorithm)

1. **Environment Setup**:
    - Load environment variables from a .env file to access sensitive keys (e.g., Groq API key).
    - Initialize NLP pipelines such as zero-shot classification, sentiment analysis, and entity extraction with Spacy and Hugging Face transformers.
2. **Contextual Memory**:
    - Create a ContextualMemory class that stores and retrieves user-specific data for better user experience over multiple interactions.
    - Extend MemorySaver to allow for storing user preferences and profiles.

3. **User Profile Management**:

    - Store and update user profiles using the UserProfile class, which records user preferences for future personalization.

4. **Feedback Collection**:

    - Collect user feedback after responses through the FeedbackCollector class, which stores the feedback for analysis.

5. **Defining the State Graph**:

    - **Intent Recognition Node**: Recognizes the user’s intent (e.g., greeting, request, feedback, or dataset loading) using the zero-shot classification model.
    - **Entity Extraction Node**: Extracts named entities from the user’s input using Spacy.
    - **Sentiment Analysis Node**: Analyzes the sentiment of the user’s input using the sentiment analysis pipeline, providing labels like “positive” or “negative” with confidence scores.
    - **Train Model Node**: Loads a dataset and trains a machine learning model (RandomForestClassifier) to classify data. The model’s accuracy is computed and stored.
    - **LLM Node**: A placeholder for large language model (LLM) generation, which is not enabled in the current version.
    - **Chatbot Logic Node**: Handles feedback requests, asking the user for feedback on the chatbot’s response.

6. **Graph Construction**:

    - **StateGraph** is built by adding the defined nodes and specifying edges (flows) between them.
    - The flow starts with intent recognition, then moves through entity extraction, sentiment analysis, model training, and LLM responses before reaching the chatbot logic node.

7. **Graph Compilation**:

    - The graph is compiled using MemorySaver to checkpoint user interactions and store conversational context.

8. **User Interaction Loop**:

    - The chatbot runs in an infinite loop, accepting user input, processing it through the graph, and generating responses.
    - Users can quit the loop by typing “quit.”

4. Code Structure

- **Environment Variables**: The code loads .env to fetch sensitive API keys and configuration.
- **NLP Pipelines**:
  - nlu_pipeline: Used for intent recognition (zero-shot classification).
  - nlp: Spacy model for named entity extraction.
  - sentiment_pipeline: Hugging Face model for sentiment analysis.
- **Memory System**:
  - ContextualMemory: Stores user-specific context during conversations.
  - UserProfile: Manages user preferences for personalization.
  - FeedbackCollector: Collects user feedback on chatbot responses.
- **State Graph**:
  - recognize_intent_node: Recognizes the intent from user input.
  - extract_entities_node: Extracts entities like names, dates, and locations.
  - analyze_sentiment_node: Analyzes the sentiment of the user's input.
  - train_model_node: Trains a Random Forest model on a dataset (loads from a specified path).
  - llm_node: Placeholder for generating responses from a large language model.
  - chatbot_logic_node: Asks for user feedback and handles the conversation flow.

5. Improvements and Recommendations

- **Dynamic Dataset Path**: Instead of hardcoding the dataset path in the train_model_node, allow users to input a file path dynamically.
- **LLM Integration**: Fully implement the LLM node to generate responses using a language model like GPT-3 or Groq.
- **Model Persistence**: Save the trained model to disk after training so that future sessions can reuse the model without retraining it each time.
- **Enhanced Error Handling**: Implement error handling for missing files or incorrect inputs during dataset loading or training.
- **Streamline Feedback Handling**: Automate feedback collection after every interaction instead of waiting for explicit feedback prompts.