LEVEL1

We figured that langgraph is used to create nodes which do specific tasks and they may be interlinked as needed.Langchain is used to build linear pipeline base workflows while langgraph is suited for complex graph based workflows. Tools are the utility that can be used by a model, it @tool is a decorator which is used to simplify the process of tool creation. Activated the google gemini api through langgraph initialised functions which Connects to the LLM (like OpenAI's GPT, Anthropic's Claude, etc.).Authenticates using API key and configures model settings — like:model name (e.g., gpt-4),temperature (controls randomness),max tokens,returns a handle or object you can use in your LangGraph workflow (as a node or tool).

NODE CREATION

Node is a function that takes in a state and returns and returns a modified state. This acts as a building block for a graph. In each node we had to invoke a prompt and response given by gemini. Then we had to keep a track of state of variables in the system so we use a library called typing and used tool typestate

LEVEL2

Weather App

To provide users realtime information based on their prompt, we use the openweather api key
Core features- user inputs a city or place, app fetches the livedata using the openweather api, displays temperature,humidity,condition,wind speed.

Challenges faced during api request without SDK

- Initialised the class with the api key and the base url for the openweather api key
- Takes a location name and returns detail

Parameters used-

- Query string
- Api key
- Unit for metric system

Making a api request-

- Sends a GET request to the API using requests.get().
- raise_for_status(): Raises an error if the HTTP request failed (e.g., 404 or 401).

Extracting Data-

Change the json file to a python dictionary which contains everything we need in output

Fashion recommender tool

No new node was used and we invoked a fix prompt before each user prompt to train the model to only respond in the field of fashion. It is used to provide clothing recommendation based on various input factors which may include the user prompt or location or both

The tool works by analyzing input data and returning clothing recommendations that are appropriate and practical. These inputs can include:

- Environmental Conditions (e.g., temperature, humidity, precipitation)
- Event Context (e.g., casual, formal, sportswear)
- User Preferences (e.g., fabric choice, color, comfort)
- Trends or Styles (optional layer of seasonal trends)

LEVEL3

Challenge-

- Gemini api models did not allow lang graph functions buffer memory to feed into its conversational capabilities

To solve this used a manual memory storage function to store the responses given by the model and invoke them as the prompt for the next user prompt

```
def build_prompt_with_history(user_input: str) -> str:
   """Construct prompt for Gemini with conversation history
included manually."""
   chat_history =
memory.load_memory_variables({})["chat_history"]
   prompt = ""
   for msg in chat_history:
       if hasattr(msg, "type"):
           if msg.type == "human":
               prompt += f"User: {msg.content}"
           elif msg.type == "ai":
               prompt += f"AI: {msg.content}”
```

We implemented conversational memory in our LangGraph workflow by adding a chat_history field to the graph's state. This memory is dynamically updated with each interaction between the user and the Gemini API. The memory retention works:

1. State Design: Includes chat_history

We defined the state to include a chat_history field — a list of message objects (dictionaries). This field tracks the conversation across turns.

2. Node Logic: Appending Memory

Each time the user sends a prompt, we:

Add the prompt to chat_history as a user message.

Send the full chat_history to the Gemini API.

Receive the Gemini response.

Append that response to chat_history as an assistant message

3. Memory Is Carried Forward

The chat_history is returned as part of the state and passed into the next node or loop, ensuring that every future prompt sent to Gemini includes full context from past turns.

LEVEL4 and SUMMARY

1)TOOL CREATION

TOOL NAME	PURPOSE
Calculator	Performs arithmetic operation
Weather tool	Returns static weather info
Fashion trends	Provides advice on fashion

2)LLM INITIALISATION
We used the ChatGoogleGenerativeAI class to integrate Google Gemini 2.0 Flash as the LLM backend.This LLM powers all the agents for answering, interpreting tools, and maintaining language understanding.
3)MULTI AGENT SETUP
We initialized 3 separate agents using LangChain's initialize_agent, each bound to one tool:

Agent Name	Tool Used	Task
Calculator agent	calculator	Math related input
Weather agent	Weather tool	Weather related input
Fashion agent	Fashion_trends	Fashion related input

4.)Memory Integration (Manual)
Instead of using LangChain's built-in memory, you manually managed memory:
memory is a list of alternating "User: ..." and "Bot: ..." strings.
On each turn:

- The entire conversation history is added to the LLM prompt.
- Gemini uses this memory for generating context-aware responses.
- This gives you full control over how memory is structured and carried forward.

5.) LangGraph Construction: StateGraph with Nodes
We created a StateGraph that models a routing and execution pipeline.

a. Defined State Schema
This schema ensures each node in the graph knows:

- What the user asked (input)
- What the agent replied (output)
- Any error
- The complete conversation so far (memory)

b. Added Nodes
Each agent is added as a LangGraph node using a make_agent_node() wrapper. This node:

- Constructs a full prompt from memory
- Sends it to the agent
- Appends the response back to memory

6.) Router Node with Conditional Logic
We created a "router" node that doesn't do anything but decides which agent to use based on the user's input
The route_logic() function inspects the user input and returns the right tool type.
7.) Start and Finish Points
router is the entry point: it's always the first step.
Each tool-agent node is a finish point: they end the graph execution.
8. Graph Execution Loop

Finally, in the REPL loop:

- You ask something like What's the weather?
- The router routes to the weather agent
- The weather agent processes it using Gemini + tool
- The response is printed and stored in memory
- The memory is retained for the next input