Level 1: Core Activation

Objective

The objective of Level 1 was to create the core "starting point" of the WeatherMind AI agent. It involved building a LangGraph node called `chatbot`, energizing it with a large language model (LLM), specifically Gemini, adding a calculator tool for numerical reasoning, and plotting the resulting computation graph to illustrate the agent's simple shape and ability.

Step-by-Step Approach

1. **Environment Preparation**
   The process began with the installation and importation of all required libraries, such as `LangGraph`, `LangSmith`, and LangChain's Gemini integration. This step ensured that the notebook environment was fully equipped to handle graph-based agent composition and LLM operations.

2. **API Configuration**
   The Gemini API key was set up in the environment variables. This was essential to provide authenticated access to Google's Gemini LLM, which serves as the conversational "brain" of WeatherMind.

3. **Integration of Calculator Tool**
   A calculator tool was created as a Python function and decorated to be detectable and callable by the LLM. It was built to compute mathematical expressions, adhering to BODMAS rules and including robust error handling for malformed input.

4. **Setting up LLM Node**
   With the tool prepared, a Gemini-powered LLM node was created and explicitly linked to the calculator tool. A custom system prompt was used to ensure the LLM employed the calculator whenever a mathematical query was detected, directing such tasks to the tool rather than solving them internally.

5. **State and Graph Construction**
   A state class was implemented to manage the conversation history based on user choice. A LangGraph `StateGraph` was constructed with the following logic:

   - The `chatbot` node handled user queries and determined whether a tool call was needed.
   - The `tools` node invoked the calculator when required.
   - Conditional routing logic ensured the chatbot either called the tool or concluded the conversation, depending on the user's response.

The graph formed a closed loop between the `chatbot` and `tools` nodes, with clear start and end points.

6. **Visualization of the Graph**
   To verify successful wiring, the compiled graph was rendered as an image. This image visually depicted the "neural pathways" of the AI, showing the flow from user input to the chatbot, to tools, and then either back or to the end node. This visualization confirmed that the architecture was implemented correctly and clearly.

7. **Testing and Verification**
   Finally, the agent was tested interactively with arithmetic queries. The chatbot correctly identified the need for computation, routed these to the calculator tool, and returned accurate answers.

Level 2: Senses of the World — Formal Report

Objective

The objective of Level 2 was to advance WeatherMind's capabilities from basic conversation and calculation and provide it with "senses"—the capacity to sense real-world information through external means. That is, specifically:

- Integrating a Weather Extractor Tool to retrieve live weather information for any given location using the OpenWeatherMap API.
- Adding a Fashion Recommender Tool to give real-time fashion trends for a queried location, with the help of Tavily or any other API.

Step-by-Step Approach

1. Selection and Rationale of Tools

For making WeatherMind context-aware and pertinent to real-world queries, I chose two tools:

- **Weather Tool**: Allows the agent to respond to questions such as "What is the weather in Mumbai?" by retrieving up-to-date weather information from an external API.
- **Fashion Trends Tool**: Enables the agent to answer questions like "What's on trend in Tokyo?" by looking up current fashion trends for a place.

These tools were selected because they comprise two very different domains—cultural trends and environmental awareness—illustrating the agent's capability to reason with and access live, external data.

2. API Configuration

I made sure that all API keys were securely configured in the environment:

- **OpenWeatherMap API Key**: Needed to retrieve live weather data.
- **Tavily API Key**: Necessary to access fashion trend search functionality (if implemented).

This step was necessary to facilitate smooth integration with live data sources and to verify the tools were operating as intended during user usage.

3. Tool Implementation

Weather Tool:
I added a function which accepts a location, makes a call to the OpenWeatherMap API, and parses the result to get and return the current temperature in Celsius. Error handling was added to handle conditions when the city cannot be found or the API is not accessible.

Fashion Tool:
I implemented a function that calls the Tavily API (or an equivalent endpoint) to obtain fashion trends for a given location. The function returns a list of trending items or an error message in case the API is down or the given location is not supported.

The two tools were properly decorated to be called by the LLM as external functions, delegating domain-specific operations from the agent core logic.

4. Graph Expansion and Integration

I modified the LangGraph structure to incorporate these new tools:

- The tool node was redesigned to encompass both the fashion and weather tools (along with the calculator of Level 1).
- The LLM node (Gemini) was re-bound to all available tools, which enabled it to choose the right tool depending on the user intent.
- The routing logic was extended such that the chatbot node could decide, depending on the user query, whether to invoke the weather tool, the fashion tool, or the calculator.

This modular graph structure guarantees extensibility in that new senses or abilities can be added in the future.

5. Visualization

Once I included the new tools, I visualized the new graph as an image.

This visualization nicely reflected the extended "neural pathways" of WeatherMind, with new limbs symbolizing the agent's access to weather and fashion information in addition to mathematical computation.

The graph verified that the system architecture was properly extended and all nodes were correctly linked.

6. Testing and Validation

I thoroughly tested the agent by making a range of queries:

- **For weather**, I queried for other cities (e.g., "What is the weather in Mumbai?"), and the agent correctly responded with live temperature information.
- **For fashion**, I queried fashion trends in other locations (e.g., "What's trending in Tokyo?"), and the agent correctly responded with appropriate current fashion information.
- I also confirmed that the calculator tool still worked as normal for arithmetic queries.
- **Error recovery** was tested by asking for non-real locations (e.g., "What is the weather in Antarctica?"), which verified the agent's graceful recovery from API errors and notification of the user.

Level 3: Judgement and Memory — Formal Report

Objective

The key objective for Level 3 was to make WeatherMind an intelligent conversation agent from a reactive tool. This involved:

- Adding routing logic so the AI could decide on its own which tool (calculator, weather, or fashion search) to invoke based on the user intent.
- Outfitting the agent with conversation memory, allowing it to recall and expand on previous conversations for more natural, context-sensitive conversation.

Step-by-Step Method

1. Tool Integration and Expansion

Upon Levels 1 and 2, I ensured the complete integration of all three tools—calculator, weather extractor, and fashion recommender—making them available to the LLM. This created a solid toolkit where the agent was capable of processing a very wide range of user questions from arithmetic problems to current weather and fashion trends.

2. Routing Logic Implementation

A key breakthrough at this point was creating intent-based routing:

- I structured the logic of the agent in such a way that when it received a user request, it would look up the intent and determine which tool to call.
- If the request was for a mathematical expression, the calculator was called; if it was weather-related, the weather tool was called; and fashion-related requests used the fashion search tool.

This routing was embedded in the LangGraph organization, and conditional edges verified that the chatbot node routed the conversation thread to the correct tool node or closed the session when required.

3. Creating Conversational Memory

To render the agent conversational and context-sensitive, I added a memory module based on LangGraph's `MemorySaver`:

- This enabled the agent to hold a record of all messages from the conversation thread.
- In doing so, the AI would be able to draw on previous topics, have continuity, and output more well-formed multi-turn conversation—much like actual human speech.

Memory was particularly valuable when following up on questions or when the user was anticipating the agent recalling information from prior interactions.

4. Graph Compilation and Visualization

Once I had combined routing and memory, I assembled the new LangGraph and visualized it as an image.

This visualization easily showed the improved structure:

- The chatbot node now had several conditional branches, each routing to a different tool depending on user intent.
- Memory was then integrated seamlessly to provide stateful, contextual conversations.

5. Comprehensive Testing and Validation

I then tested the agent with an assorted group of questions to confirm routing as well as memory:

- **Math questions** were properly routed to the calculator, with correct answers returned.

- **Weather-related questions** activated the weather tool, retrieving live information and reporting errors gracefully in case of unrecognized places.
- **Fashion-related questions** called the fashion recommender, which returned the latest trend information.