Report of Building an intelligent Travel Assistant AI

🧠 How LLM Was Used for Reasoning

i. LLM-Based Reasoning Process

In this notebook, we use Google's **Gemini-Pro** large language model (LLM) via the LangChain framework. The reasoning process works through the **Tool-Calling Agent** architecture.

The flow is as follows:

1. **User Query**: A natural language question is given to the system — e.g., "Tell me the weather and top attractions in Coimbatore."

2. LLM Decision Making:

  * The LLM parses the query.

  * It decides which tools are needed to satisfy the query based on function descriptions.

  * For example, it may call `get_weather` and then `top_attractions` using LangChain's tool-calling capability.

3. Tool Execution:

  * `get_weather` calls the Weather API.

  * `top_attractions` uses Tavily Search API.

4. Final Reasoning:

  * The LLM aggregates the tool responses.

  * It composes a human-readable answer like:

*"The weather in Coimbatore is sunny with 34°C... Top attractions include..."

* The reasoning step combines raw outputs with contextual phrasing, summarization, and formatting.

This LLM reasoning mimics how a human assistant would synthesize multiple sources of information to answer a multi-part query.

🧾 Code Explanation and Program Flow

ii. Program Flow Overview

The notebook is divided into modular steps:

1. Install Dependencies

```python
!pip install -q langchain ... tavily-python requests pdfkit
```

Installs necessary libraries like `langchain`, `tavily-python`, and `pdfkit`.

2. Import Libraries

```python
from langchain_google_genai import ChatGoogleGenerativeAI
...
```

Loads LLMs, tools, and agent helpers from LangChain.

3. API Key Setup

```python
os.environ['GOOGLE_API_KEY'] = "..."
```

Stores API keys for Gemini, Tavily, and WeatherAPI.

4. Define Tools

* **`get_weather(location)`**:

  Calls the WeatherAPI and returns current temperature, humidity, and conditions.

* **`top_attractions(city)`**:

  Uses Tavily’s search engine to fetch top tourist spots for a given city.

5. Create LangChain Agent

```python
agent = create_tool_calling_agent(...)
```

This agent routes the user's query to appropriate tools, then composes the final output using Gemini-Pro.

6. Run Query

```python
response = agent_executor.invoke({"input": f"...Coimbatore..."})
```

Invokes the agent to get weather and attractions in a single unified response.

7. Display & Export Output

```python
display_styled_response(response["output"])

save_response_as_pdf(...)
```

* Styled Output: Visually displays the result in a styled HTML card.

* PDF Export: Saves the styled HTML as a PDF using `pdfkit`.

This structure makes the notebook easily extendable. You can plug in new tools (e.g., hotel search, flight data) with minimal changes to the reasoning flow.