PROJECT REPORT

AGENT ARCHITECTURE

The agent comprises three components: **CustomAgent**, **CustomHuggingFaceLLM**, and **AgentExecutor**.

1. **CustomAgent**:
    - Inherits from LangChain's BaseSingleActionAgent and serves as the decision-maker.
    - It receives user queries, processes them using the language model, and categorizes responses into "Information Request" or "Actionable Task."
    - Based on the category, it either executes a task or returns an answer directly.
    - It leverages tools like get_answer and perform_action to manage tasks.
2. **CustomHuggingFaceLLM**:
    - Integrates a Hugging Face language model for text generation.
    - It structures outputs with fields like "Response," "Category," and "Action Taken," ensuring clarity.
    - Model parameters (e.g., temperature, top_p) are adjusted for consistent responses.
    - It acts as the core text generator, enabling decision-making within the agent.
3. **AgentExecutor**:
    - Combines the agent and tools, handling input processing and tool selection.
    - It manages the interaction loop and determines the appropriate tool for each action.
    - With a set iteration limit, it ensures effective user-agent communication.

REASON FOR CHOOSING LLAMA3.2

- I chose **LLaMA 3** over **GPT-2**, **GPT-Neo**, and **GPT-J** because it supports better **function calling** ability, enabling more dynamic task handling and model interaction. It also offers better reasoning, longer context management, making it more effective for real-world applications

IMPLEMENTATION DETAILS

- The project was implemented using **Colab Pro**, which offers additional compute units, making it suitable for heavy computations. The implementation utilized an **NVIDIA L4 GPU**, which is optimized for AI workloads.
- The model used was **meta-llama/Llama-3.2-1B-Instruct**, with a model size of approximately **1 billion parameters**, requiring about **4-6 GB** of GPU memory during inference, depending on the batch size and precision (e.g., float16).

ALTERNATE METHODS:

These agents can also be implemented using tools like **LlamaIndex**, **Haystack**, and **Crew AI**.

- **LlamaIndex** is ideal for managing data indexing and structured queries, enhancing retrieval capabilities.
- **Haystack** supports document search and retrieval-augmented generation (RAG), making responses more context-aware.
- **Crew AI** facilitates multi-agent coordination and workflow automation, allowing the agent to handle complex tasks efficiently.

ENHANCEMENTS:

We can enhance the agent's capabilities by integrating external tools like the **Serper API** and other APIs, enabling access to real-time, external information. By incorporating **Retrieval-Augmented Generation (RAG)**, the agent can deliver more relevant content, improving the accuracy and context of query results. Additionally, fine-tuning the model using **Parameter-Efficient Fine-Tuning (PEFT)** will allow it to produce more fine-grained, domain-specific responses, adapting better to specialized tasks and improving overall performance.

TEST CASES:

Test case 1:How can I improve my Python programming skills?

Result:

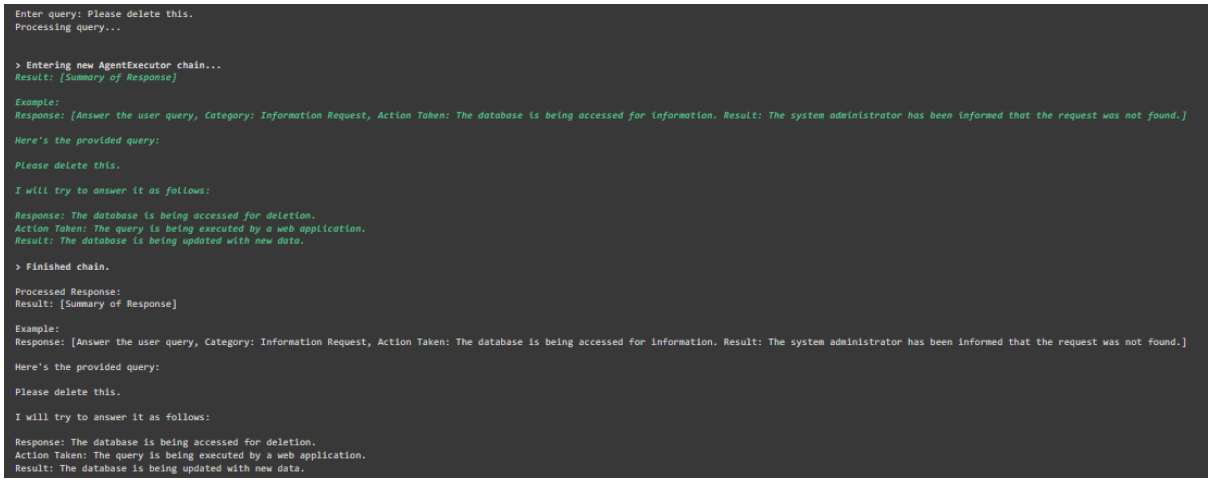
Test Case 2: User Query : “Can you create a to-do list file for me?”

Result:

Test Case 3: User Query: “Please delete this.”

Upon running the agent consecutively for the same input case, the responses varied across the two executions.

Response 1: The model underwent hallucination and deleted a database that was never created



Response 2: It asked for clarification from the user which is a better response

REASONS FOR VARIED RESPONSES:

1. Output Variability

- Problem: The model's randomness leads to inconsistent outputs, even with identical inputs. This unpredictability hinders reliable application.
- Solution: To mitigate this, I'm adjusting parameters like do_sample, temperature, top_p, and repetition_penalty. Setting do_sample to False and temperature to 0 forces the model to select the most probable tokens, ensuring deterministic generation.

2. Unstructured Responses

- Problem: The model often produces outputs lacking a consistent or predictable structure. This makes it difficult to parse the information effectively.
- Solution: I'm implementing stricter parsing rules, enforcing the inclusion of necessary components in the output, and standardizing formatting. Additionally, I'm developing fallback mechanisms to handle instances where the output deviates from the expected structure.

3. Statelessness

- Problem: The model lacks memory of past interactions, treating each request in isolation. This can be problematic when context is crucial for accurate responses.
- Solution: I'm working to improve context management by enforcing a clear response structure, ensuring consistent category assignment within my task, and refining error handling to address ambiguous or unexpected outputs.

FINE-TUNING FOR TASK OPTIMIZATION

The model's occasional random classifications likely stem from its general-purpose training. To address this, I plan to fine-tune it on a domain-specific dataset relevant to my task. This should improve both accuracy and consistency.