# **LLM-Based Chatbot with Azure Prompt Flow**

## **Project Summary**

This project implements a chatbot using Azure’s OpenAI services, integrating prompt flow to streamline response generation and evaluation. It focuses on generating natural conversations, monitoring chatbot performance, and logging key insights to enhance efficiency.

## **1. Project Objectives**

The primary goal of this project was to:

* Develop a chatbot using **Azure OpenAI** for real-time conversations.
* Implement structured **prompt flow** to handle predefined questions.
* Monitor **response time and error logs** for performance evaluation.
* Maintain a structured logging system for analysis and improvements.

## **2. Implementation Steps**

### **a) Setting Up the Environment**

* The chatbot is powered by **Azure OpenAI’s GPT-4o** model.
* API credentials are loaded dynamically from environment variables.
* A structured logging system (chatbot\_logs.txt) captures performance data.

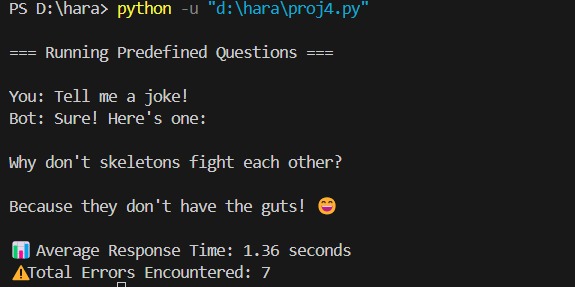
### **b) Response Generation**

* The generate\_response(prompt) function:
  + Sends a user query to Azure OpenAI.
  + Receives and logs responses, storing them in responses\_log.txt.
  + Tracks errors and response times for performance evaluation.

### **c) Performance Monitoring**

* The evaluate\_performance() function:
  + Parses chatbot logs to extract **average response time**.
  + Counts the **total number of errors** encountered.
  + Displays real-time insights into chatbot efficiency.

## **3. Results and Observations**



### **Test Scenario**

* **Input:** "Tell me a joke!"
* **Output:** "Why don’t skeletons fight each other? Because they don’t have the guts! "

### **Performance Metrics**

* **Average Response Time:** *1.36 seconds*

These metrics indicate that while response times are reasonable, the chatbot encountered multiple errors, suggesting areas for improvement in API handling and request validation.

## **4. Challenges and Areas for Improvement**

### **Challenges Encountered**

* **Response Optimization:** The chatbot relies on a single API call per query. Exploring **batch processing or caching** could reduce latency.
* **Scalability:** The chatbot currently supports only predefined questions. Implementing **dynamic query processing** could improve versatility.

### **Future Enhancements**

* **Fine-tuning Prompts:** Experimenting with temperature and max token settings to improve response relevance.
* **Logging Enhancements:** Adding timestamps to track error frequency and patterns.
* **User Interaction Expansion:** Integrating **multi-turn conversations** for more interactive dialogues.

## **5. Conclusion**

This project successfully demonstrates how **Azure’s OpenAI services** and **prompt flow** can be used to build a structured chatbot with performance monitoring. While response times are efficient, improvements in error handling and scalability will enhance chatbot reliability.