**Al-Driven Guest Experience Personalization System for Hospitality**

**Milestone 2 Documentation:**

**Objective :**

The goal is to build a real-time sentiment analysis and service alert system. This system uses advanced language models such as OpenAI GPT or Meta LLaMA to analyze guest feedback across various touchpoints. Alerts are generated for negative sentiments or potential service issues, helping improve customer satisfaction and operational efficiency.

**Table of Contents :**

1. Overview
2. Setting Up the Development Environment
3. Required Libraries and Extensions
4. Backend Implementation
5. Sentiment Analysis Using LLMs
6. Alert Generation
7. Building the app.py File
8. Running the Streamlit App in Google Colab
9. Testing and Validating the App
10. Conclusion
11. **Overview**

This document explains how to create a backend and frontend for a sentiment analysis and service alert engine using Streamlit and large language models (LLMs). The app will analyze real-time feedback, compute sentiment scores, and generate alerts for potential service issues.

1. **Setting Up the Development Environment**

I preferred google colab to develop this project.

And this is the step by step process includes.

Step 1: Open Google Colab

Google Colab provides a free and powerful cloud-based development environment suitable for Python-based projects.

Step 2: Install Required Libraries

We need to install the following libraries in Colab:

Streamlit: For creating an interactive web interface.

Pyngrok: To expose the Streamlit app to the internet.

Transformers: For working with LLMs such as GPT or Meta LLaMA.

Run the following commands in Colab:

Python

Copy code

!pip install streamlit pyngrok transformers

1. **Required Libraries and Extensions**

This are the libraries used in our project and done sentiment analysis

Correctly

Here is an overview of the libraries used and their purposes:

**Streamlit :**

Purpose: Create a user-friendly interface for real-time sentiment analysis.

Installation: !pip install streamlit

Key Features: Input fields, dynamic updates, and user interaction.

**Pyngrok :**

Purpose: Expose the local Streamlit server to the internet for external access.

Installation: !pip install pyngrok

Key Features: Secure tunneling for localhost servers.

**Transformers :**

Purpose: Use pre-trained LLMs (OpenAI GPT or Meta LLaMA) for natural language processing tasks.

Installation: !pip install transformers

Key Features: Provides APIs for deploying state-of-the-art language modelling.

**How They Work Together ?**

Streamlit is the frontend framework used to design the application interface, allowing users to input feedback and view results dynamically.

Pyngrok bridges the gap between the local Streamlit server and the internet, creating a secure public link for the application.

Transformers powers the backend sentiment analysis engine, utilizing pre-trained models to analyze the user’s input and generate insights like sentiment, confidence, and alerts.

**4. Backend Implementation (Extended)**

The backend of the system is designed to process user feedback, analyze it for sentiment, and generate alerts for service issues. This involves three main steps: loading the pre-trained sentiment analysis model, processing feedback, and generating alerts. Here’s a detailed breakdown:

Step 1: Load Pre-trained Model

We use the Hugging Face `pipeline` API to load a pre-trained sentiment analysis model. These models are trained on large datasets and can classify feedback as positive, negative, or neutral with confidence scores. For simplicity, we use the DistilBERT model fine-tuned for sentiment analysis.

Code:

```python

From transformers import pipeline

# Load the sentiment analysis model

Sentiment\_pipeline = pipeline(“sentiment-analysis”)

```

Explanation:

- The `pipeline(“sentiment-analysis”)` loads a pre-trained model optimized for text sentiment classification.

- The model works out-of-the-box, requiring no additional training or setup.

- It returns predictions in the form of a label (`POSITIVE` or `NEGATIVE`) and a confidence score between 0 and 1.

Step 2: Process Guest Feedback

In this step, the feedback provided by the user is analyzed to determine its sentiment and the associated confidence level. This analysis enables the system to evaluate the tone of the feedback.

Code:

```python

Def analyze\_feedback(feedback):

# Get sentiment analysis result

Result = sentiment\_pipeline(feedback)

Sentiment = result[0][“label”] # Extract the sentiment label

Confidence = result[0][“score”] # Extract the confidence score

Return sentiment, confidence

```

Explanation:

- The `feedback` string provided by the user is passed to the sentiment analysis model.

- The model returns a dictionary containing:

- `label`: The predicted sentiment (`POSITIVE` or `NEGATIVE`).

- `score`: The confidence score (probability of the prediction being correct).

- This information is extracted and returned for further processing.

Step 3: Alert Generation

To ensure proactive service improvement, alerts are generated for feedback that is either:

- Negative in sentiment.

- Has a confidence score below a defined threshold (e.g., 0.5)

Code:

```python

Def generate\_alert(sentiment, confidence):

If sentiment == “NEGATIVE” or confidence < 0.5:

Return “Alert: Potential service issue detected!”

Return “No alerts.”

``

Explanation:

- Feedback with a sentiment of “NEGATIVE” is flagged as a potential issue.

- If the model’s confidence score is below 50%, it indicates uncertainty in prediction, and an alert is also raised.

- This ensures that even ambiguous feedback is reviewed for service improvement opportunities.

Example:

1. Feedback: “The staff was unprofessional.”

- Sentiment: `NEGATIVE`

- Confidence: `0.92`

- Alert: “Alert: Potential service issue detected!”

2. Feedback: “The food was okay.”

- Sentiment: `POSITIVE`

- Confidence: `0.48`

- Alert: “Alert: Potential service issue detected!”

1. **Building the app.py File**

For backend purpose

The app.py file is the heart of the Streamlit application. It defines the user interface and backend interactions.

Code for app.py

Python

Copy code

Import streamlit as st

From transformers import pipeline

# Load sentiment analysis model

Sentiment\_pipeline = pipeline(“sentiment-analysis”)

# Streamlit app title

St.title(“Sentiment Analysis & Service Alert Engine”)

# User input for feedback

Feedback = st.text\_input(“Enter guest feedback:”)

If feedback:

# Analyze sentiment

Sentiment, confidence = analyze\_feedback(feedback)

St.write(f”Sentiment: {sentiment}”)

St.write(f”Confidence: {confidence:.2f}”)

# Generate alert

Alert = generate\_alert(sentiment, confidence)

St.write(alert)

**How the app.py File Works Together ?**

User Interaction:

The app prompts the user to enter feedback through a text input field.

Processing:

The entered feedback is sent to the backend for sentiment analysis using the analyze\_feedback() function.

Sentiment and confidence scores are calculated and returned.

Output:

The sentiment and confidence score are displayed on the app interface.

Based on the analysis, an alert is generated (if needed) and displayed.

Real-Time Updates:

Streamlit automatically refreshes the interface whenever the user inputs new feedback, providing real-time results.

1. **Running the Streamlit App in Google Colab**

In Google colab set up this and signup in ngrok and create authentication key

Step 1: Set Up pyngrok

To make the app accessible online, set up a secure tunnel using ngrok.

Python

From pyngrok import ngrok

# Set your ngrok authtoken

Ngrok.set\_auth\_token(“your\_ngrok\_token\_here”)

# Open a tunnel on port 8501

Public\_url = ngrok.connect(8501)

Print(f”Streamlit app is running at: {public\_url}”)

Step 2: Run the App

Run the app.py file to start the Streamlit server.

In cmd prompt type this :

!streamlit run app.py

And you will see the streamlit app ngrok url with authentication.

1. **Testing and Validating the App**

Open the public URL provided by ngrok.

Enter guest feedback into the input box.

The app will display:

Sentiment: Positive/Negative/Neutral.

Confidence: A score representing model certainty.

Alerts: If negative sentiment or low confidence is detected.

**Common Scenarios for Testing**

Positive Sentiment:

Feedback: “The bed was very comfortable.”

Expected: Sentiment: Positive, Confidence: High

Negative Sentiment:

Feedback: “The staff was rude.”

Expected: Sentiment: Negative, Alert Generated

Neutral Feedback:

Feedback: “The hotel is located near the city center.”

Expected: Sentiment: Neutral, Confidence: Moderate

Ambiguous Feedback:

Feedback: “The food was acceptable, but nothing special.”

Expected: Sentiment: Neutral, Confidence: Low, Alert Generated

1. **Conclusion**

In this document, we built a sentiment analysis and service alert engine using Streamlit, Hugging Face models, and ngrok. The app provides an intuitive interface to analyze guest feedback, compute sentiment, and generate alerts for service issues.