PROJECT REPORT ON

Title - [“Building a Negotiation Chatbot Using a Pre-trained AI Model - Google Gemini AI”]

by

Vijay Gurung

Submitted To – [“Settyl”, Chennai,India]

Demo Video URL :

[https://drive.google.com/file/d/1tmvOwivz7qxAdiz3b-LdXtneGg11_GCK/view?usp=sharing]

TABLE OF CONTENTS

ABSTRACT

This project focuses on building an AI-powered **"Negotiation Chatbot"** that simulates a realistic negotiation process between a customer and a supplier. The chatbot leverages pre-trained AI models, including the Gemini API, to conduct dynamic conversations, making price offers, accepting, rejecting, or countering customer proposals. The chatbot integrates sentiment analysis to adjust its negotiation strategy based on the user's tone offering more favourable deals if the user is polite or positive in their approach.

The project is designed using Python, incorporating the **NLTK library** for sentiment analysis and **Streamlit** for a user-friendly web interface. The chatbot employs pricing logic, allowing flexibility within a specified price range and provides responses tailored to the user's input, simulating a seamless, automated negotiation experience.

The application is aimed at industries such as retail and e-commerce, where customer-supplier interactions are frequent. It can be integrated into platforms to enhance customer engagement and optimize pricing strategies.

Keywords: Negotiation Chatbot, AI, Sentiment Analysis, Gemini API, Python, Streamlit, E-commerce, Price Negotiation, Web Interface.

Introduction

This project showcases the development of a sophisticated **negotiation chatbot**, designed to simulate real-life price negotiations between a customer and a supplier. The chatbot is powered by an advanced AI model, such as **Google Gemini**, enabling it to respond intelligently and dynamically based on user inputs. This includes handling key negotiation aspects like price offers, discount requests and sentiment-based adjustments to the conversation tone.

The chatbot is built with **Streamlit**, which provides a smooth and interactive user interface, allowing users to easily engage with the chatbot. The **requests** library is used to send **HTTP requests** and seamlessly communicate with the AI model's API, enabling real-time exchanges. Additionally, the **NLTK (Natural Language Toolkit)** library plays a crucial role in performing sentiment analysis, allowing the chatbot to adapt its responses based on the emotional context of the user's input becoming more empathetic, firm or flexible as the situation demands.

The negotiation process is structured around pricing logic, where users can input price offers and receive counteroffers generated by the AI. This ensures a fluid interaction, where the chatbot mimics realistic negotiation tactics, such as offering discounts or rejecting offers based on the perceived sentiment and context of the conversation. Overall, this project illustrates how AI can be utilized to automate and enhance the negotiation process, providing an engaging and efficient experience for both sides.

Project Overview

The negotiation chatbot offers a dynamic platform for users to engage in simulated price negotiations, mirroring real-world interactions between buyers and sellers. Here's a detailed breakdown of its capabilities:

- **Propose a Price for a Product:** Users can initiate the negotiation process by submitting their desired price for a product. This feature allows for a realistic negotiation experience, where the user's offer is the starting point for the discussion.
- **Receive Counteroffers or Discounts:** Based on the initial proposal, the chatbot utilizes its internal pricing logic to generate counteroffers or discount suggestions. This functionality is designed to reflect typical negotiation strategies, such as suggesting a reduced price or providing rationale for the price adjustment. The AI model's decision-making process is informed by predefined rules or algorithms that simulate how a supplier might respond in a real negotiation scenario.
- **Engage in Multiple Negotiation Cycles:** The chatbot supports a series of negotiation cycles, allowing users to refine their offers and responses based on ongoing interactions. This iterative process is facilitated by sentiment analysis from the NLTK library, which helps the chatbot gauge the user's emotional tone and adjust its responses accordingly. For example, if a user becomes frustrated, the chatbot might offer more favorable terms or alter its approach to de-escalate the situation.

The integration of a pre-trained AI model, such as Google Gemini AI is central to the chatbot's conversational abilities. These models are equipped with advanced natural language processing capabilities, enabling them to understand and generate human-like responses. This ensures that the chatbot can handle complex conversational scenarios, maintain context over multiple exchanges, and provide a more engaging and realistic negotiation experience. The AI model's role is crucial in crafting responses that are not only contextually appropriate but also strategically aligned with negotiation principles.

Technology Used in the Negotiation Chatbot Project

The **Negotiation Chatbot Project** incorporates various technologies to ensure seamless interaction, processing of user input, and effective communication with AI models.

Below is an overview of the technologies used:

1. **Streamlit:**
    - **Role:** Utilized to build an intuitive and interactive user interface for the chatbot. Streamlit allows users to interact with the chatbot in real-time, providing a web-based environment for input and displaying results.
2. **Requests:**
    - **Role:** Used to send HTTP requests to the AI model's API - Google Gemini. This facilitates communication between the chatbot's backend and the AI model, enabling real-time conversational responses.
3. **NLTK (Natural Language Toolkit):**
    - **Role:** Employed to perform sentiment analysis on the user's input. NLTK helps in analyzing and understanding the emotional tone behind the user's message, which can guide the chatbot's negotiation strategy.
4. **Gemini API:**
    - **Role:** The core AI engine that handles conversational responses and negotiations. The API processes user inputs, generates context-aware responses, and provides intelligent suggestions during the negotiation process.

This technology stack ensures that the Negotiation Chatbot is both interactive and capable of handling complex conversations, making it an effective tool for negotiation tasks.

Model Integration and Setup

Model Integration and Setup involves several critical steps to ensure that the AI model operates seamlessly within the chatbot framework. This section outlines the process of integrating a pre-trained AI model, configuring it for negotiation tasks, and setting it up to interact effectively with the user interface.

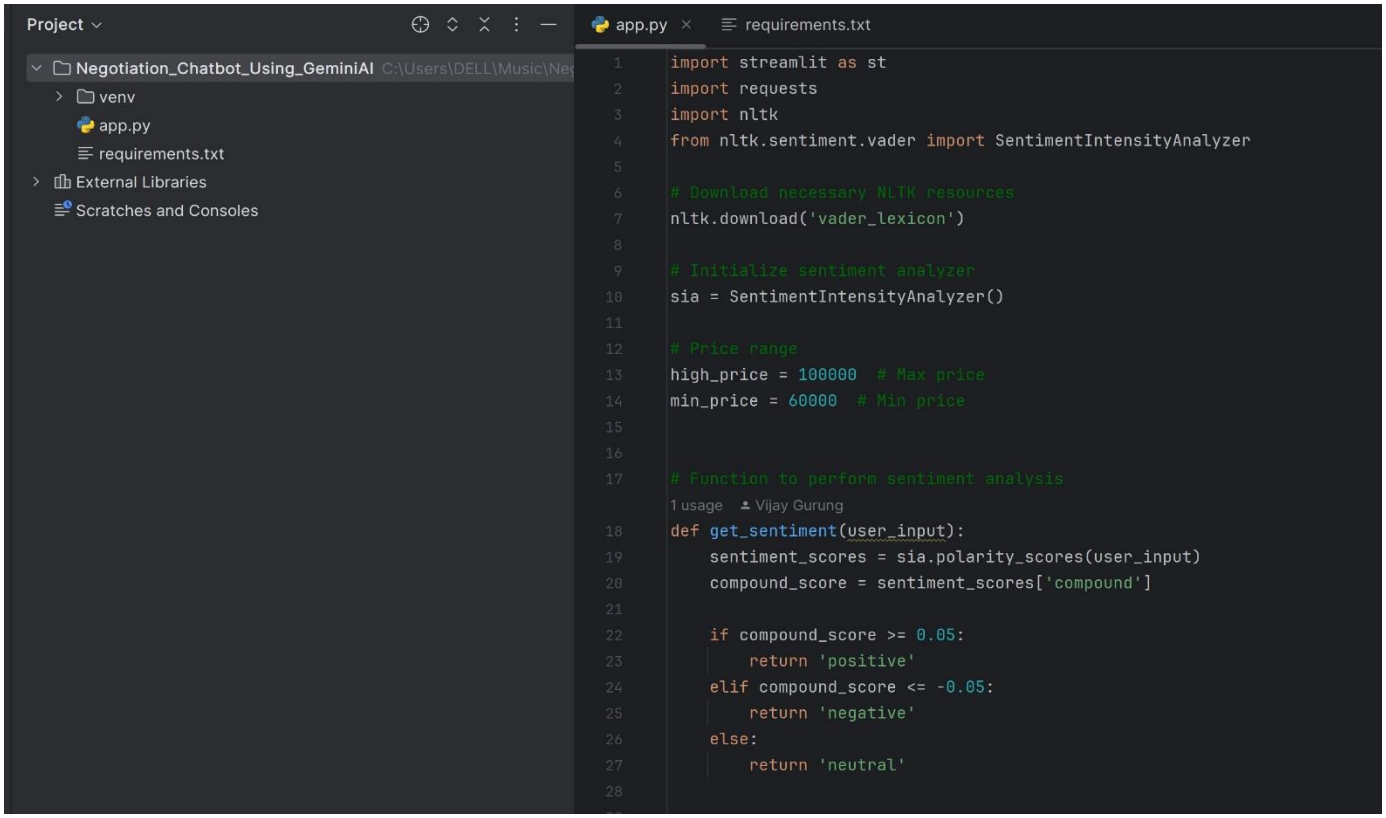
1. **Choosing the AI Model:**
    - **Selection Criteria:** Choose a pre-trained AI model such as Google Gemini based on its capabilities in handling conversational interactions and understanding context. The model should be well-suited for generating dynamic responses and managing negotiation scenarios.
    - **Model API:** Ensure that the chosen model has a well-documented API for easy integration. This API will be used to send and receive data between the chatbot application and the AI model.
2. **Setting Up the API Integration:**
    - **API Key and Authentication**: Obtain the necessary API key or credentials for accessing the AI model's API. This typically involves signing up for the service and configuring authentication settings.
    - **Requests Library:** Use the Requests library in Python to handle HTTP requests to the AI model's API. This involves sending user inputs to the model and receiving responses for further processing.
    - **Configuration:** Set up the API endpoint and configure the parameters required for the API requests, such as model type, prompt format and response settings.
3. **Configuring the Model for Negotiation:**
    - **Prompt Engineering:** Design and fine-tune prompts that guide the AI model in generating responses relevant to price negotiations. This may involve creating templates or example dialogues that reflect typical negotiation scenarios.
    - **Internal Pricing Logic:** Implement internal logic to manage pricing adjustments, counteroffers, and discount calculations. This logic can be encoded within the chatbot application to complement the model's responses.
    - **Sentiment Analysis Integration:** Incorporate NLTK for sentiment analysis to adjust the model's responses based on the user's emotional tone. This step ensures that the chatbot's replies are sensitive to user sentiment and improve the overall negotiation experience.

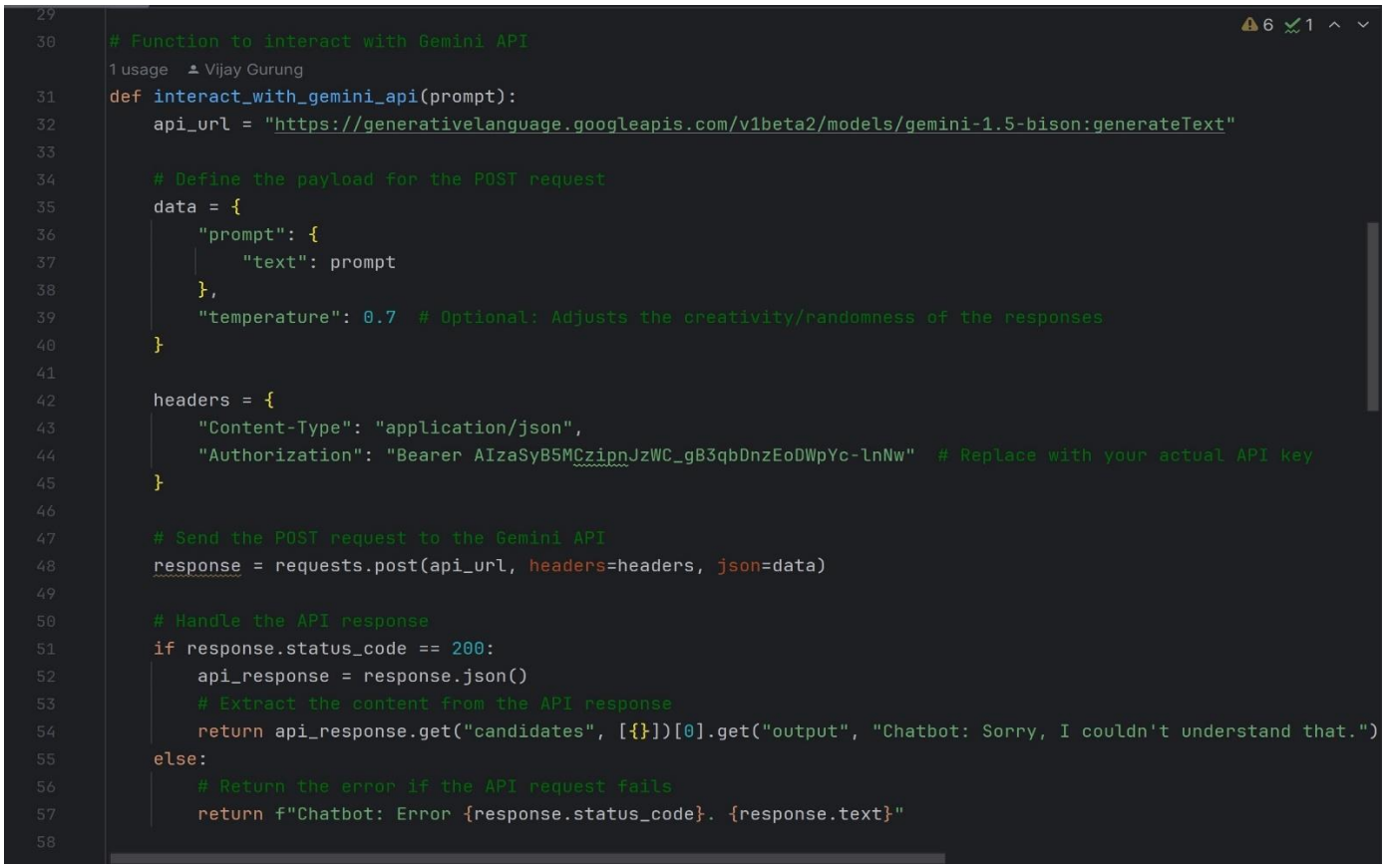
4. **Testing and Validation:**

   - **Scenario Testing:** Conduct thorough testing with various negotiation scenarios to validate the model's performance. This includes checking the accuracy of responses, handling edge cases, and ensuring the chatbot meets the desired interaction quality.
   - **Performance Monitoring:** Monitor the chatbot's performance to identify any issues or areas for improvement. This includes tracking API response times, accuracy of negotiation logic, and user satisfaction.

By following these steps, the AI model can be effectively integrated into the chatbot framework, providing a robust foundation for simulating price negotiations and enhancing user interactions.

Code Snippet





app.py × requirements.txt

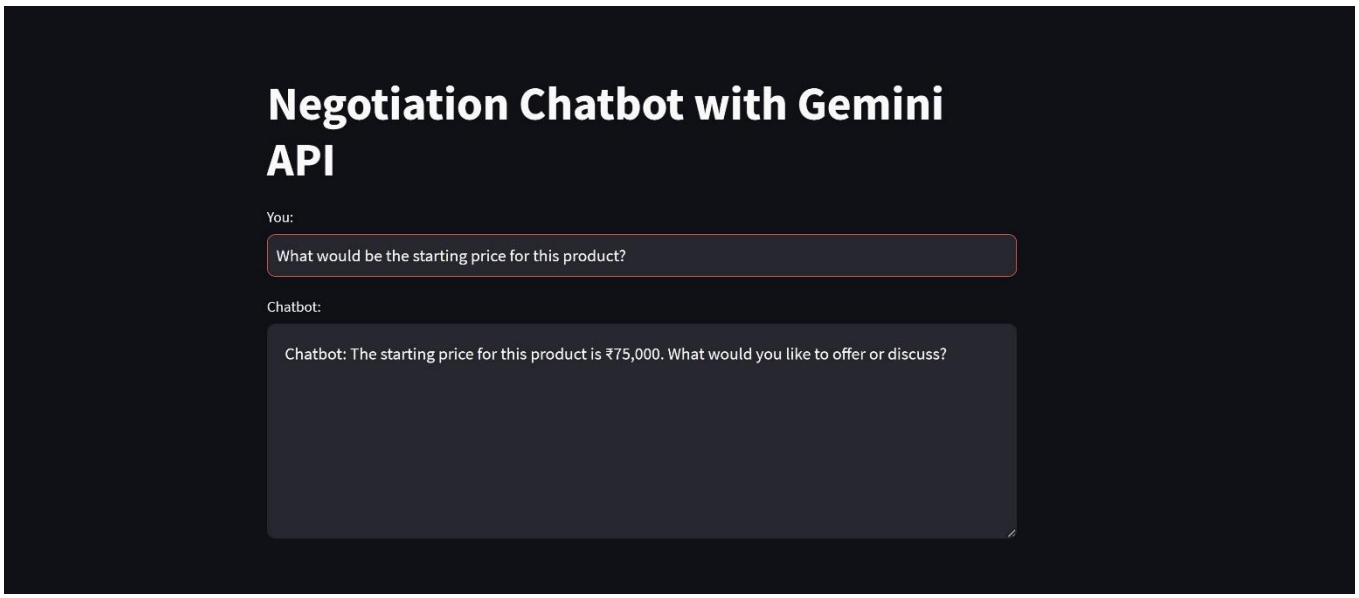
```python
# Function to extract price from user input
1 usage  Vijay Gurung
def extract_price(user_input):
    words = user_input.split()
    for word in words:
        if word.startswith('₹') and word[1:].isdigit():
            return int(word[1:])
        elif word.isdigit():
            return int(word)
    return None


# Function to simulate negotiation
1 usage  Vijay Gurung
def negotiate(user_input):
    sentiment = get_sentiment(user_input)

    if "starting price" in user_input.lower():
        return "Chatbot: The starting price for this product is ₹75,000. What would you like to offer or discuss?"

    if "discount" in user_input.lower():
        return "Chatbot: I'm glad you're interested! We're currently discussing a product with a price range betwee

    if "best price" in user_input.lower():
        if sentiment == 'positive':
            return "Chatbot: Since you're interested, I can offer you a discount. How about ₹70,000?"
        else:
            return "Chatbot: I understand you're asking for a discount. I can offer ₹75,000."

    if "feel" in user_input.lower() and "too high" in user_input.lower():
        return "Chatbot: I understand ₹70,000 might seem high. How about we settle at ₹65,000?"
```

```python

    if "negotiate" in user_input.lower() or "meet halfway" in user_input.lower():
        offer_price = extract_price(user_input)
        if offer_price:
            if offer_price >= high_price:
                return "Chatbot: I appreciate your offer, but ₹75,000 is the best I can do."
            elif offer_price >= min_price:
                return f"Chatbot: We could meet at ₹{offer_price + 5000}. What do you think?"
            else:
                return f"Chatbot: ₹{offer_price} is too low. The minimum price I can offer is ₹{min_price}."
        else:
            return "Chatbot: Could you please specify your offer?"

    if "settle" in user_input.lower() or "negotiation" in user_input.lower():
        return interact_with_gemini_api(user_input)

    if "interested" in user_input.lower() and sentiment == 'positive':
        return "Chatbot: I'm glad to hear you're interested! Since you're being positive, I can offer you a special

    if "concern" in user_input.lower() and sentiment == 'negative':
        return "Chatbot: I understand your concern. I can offer a price of ₹70,000, which is the best I can do."

    return "Chatbot: How can I assist you further with the pricing?"
```

```python

# Streamlit App
st.title("Negotiation Chatbot with Gemini API")

# Get user input
user_input = st.text_input("You:", "What would you like to negotiate about?")

# Define chatbot response
if user_input:
    response = negotiate(user_input)
    st.text_area("Chatbot:", value=response, height=200)

```

Outcome

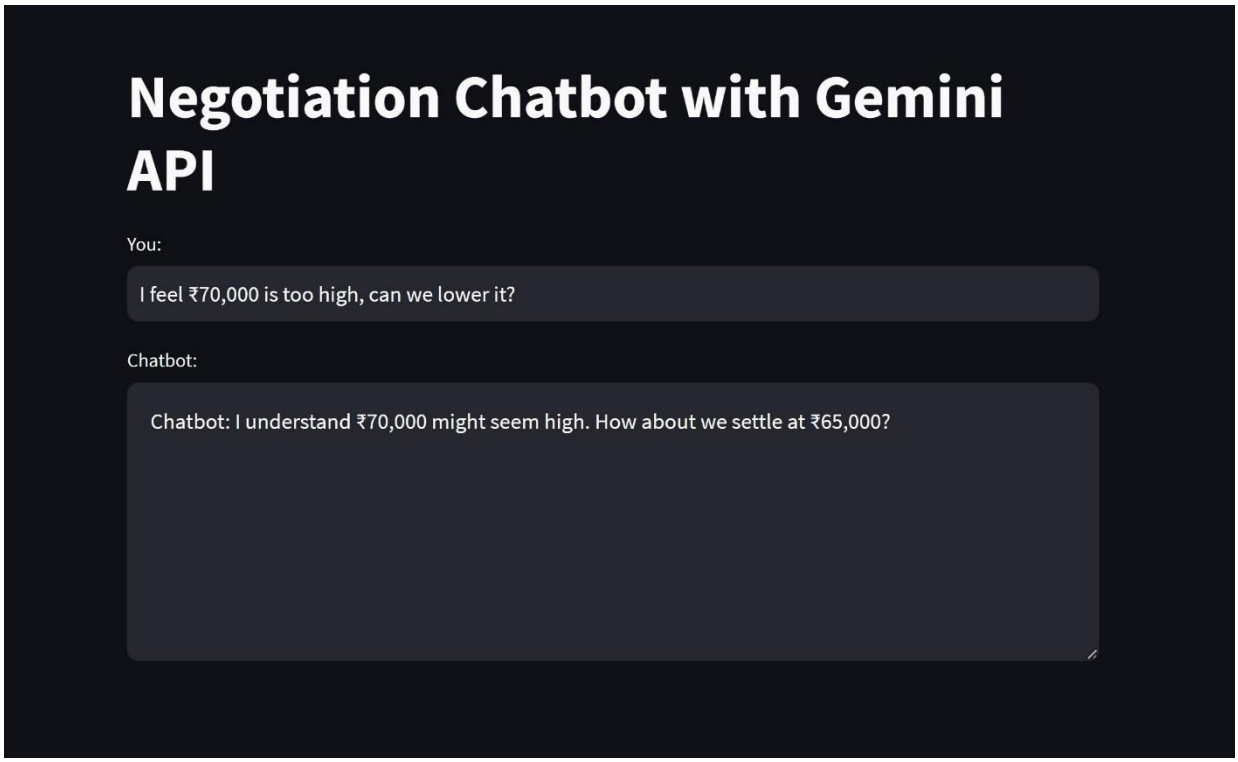
Question No.1 : **What would be the starting price for this product?**



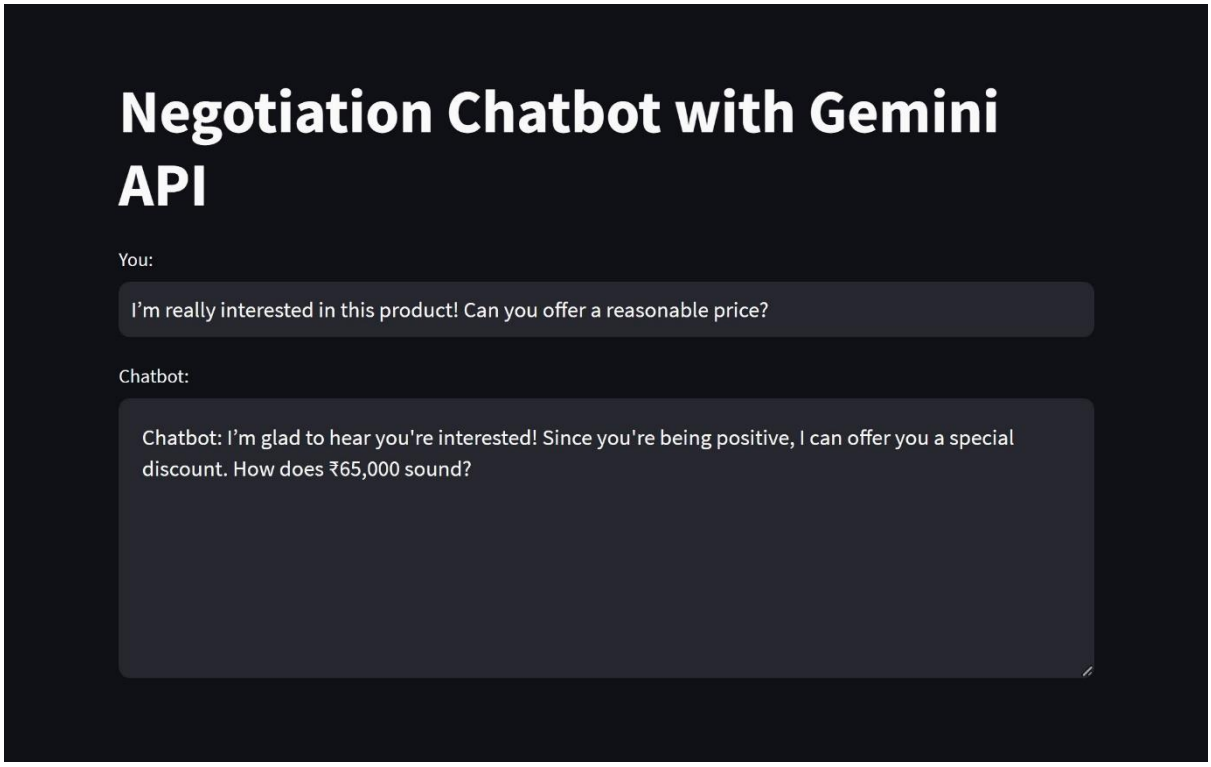
Question No.2 : **I am looking for a discount, what can you offer?**

Question No.3 : **What's the best price you can give me without going too high?**

Question No.4 : **I feel ₹70,000 is too high, can we lower it ?**



Question No.5 : **I’m really interested in this product! Can you offer a reasonable price?**



Chatbot Logic Implementation

Chatbot Logic Implementation is a crucial part of developing a negotiation chatbot, as it involves designing and coding the core functionalities that drive the negotiation process. This section details the steps involved in implementing the logic that governs how the chatbot interacts with users, processes their inputs and manages the negotiation flow.

1. **User Input Handling:**
    - **Input Reception**: The chatbot must capture user inputs effectively, including proposed prices, requests for discounts, or any other negotiation-related queries. This is achieved through the Streamlit interface, where user inputs are collected via text fields or other interactive elements.
    - **Preprocessing:** Clean and preprocess the user inputs to ensure they are in a format suitable for processing. This may involve tasks such as removing unwanted characters, normalizing text, and extracting relevant information.
2. **Negotiation Logic:**
    - **Pricing Logic:** Implement the core negotiation logic that manages price proposals, counteroffers, and discount calculations. This logic should handle various negotiation scenarios, such as:
        - **Initial Offer:** When a user proposes a price, the chatbot evaluates it based on predefined criteria For example - minimum acceptable price, maximum discount).
        - **Counteroffer Generation:** Based on the initial offer, the chatbot generates a counteroffer or discount offer. This could be a percentage discount or a fixed price adjustment.
        - **Offer Acceptance/Rejection:** The chatbot determines whether the user's response (acceptance or further negotiation) is valid and updates the negotiation status accordingly.
3. **AI Model Integration:**
    - **API Interaction:** Use the Requests library to send user inputs to the AI model and receive responses. The AI model's responses should be integrated into the negotiation logic to provide intelligent and contextually relevant replies.
    - **Response Parsing:** Extract and interpret the AI model's responses to generate appropriate chatbot actions, such as presenting counteroffers, acknowledging user requests, or suggesting alternative solutions.

4. **Sentiment Analysis:**

   - **Sentiment Detection:** Apply NLTK to analyze the sentiment of user inputs. This helps the chatbot adjust its responses based on the user's emotional tone, enhancing the interaction experience.
   - **Adaptive Responses:** Modify the chatbot's responses based on sentiment analysis results. For example, if a user's input indicates frustration, the chatbot might offer a more empathetic reply or better terms.

5. **Dialogue Management:**

   - **Context Management:** Maintain the context of the conversation throughout the negotiation process. This involves tracking previous offers, user preferences, and negotiation history to provide coherent and relevant responses.
   - **Flow Control:** Implement logic to handle various stages of the negotiation process, such as initiating the negotiation, making offers, and concluding the negotiation based on user inputs and responses.

6. **Error Handling and Edge Cases:**

   - **Error Handling:** Incorporate mechanisms to handle errors and unexpected inputs. This includes validating user inputs, managing API errors, and providing fallback responses in case of issues.
   - **Edge Case Management**: Plan for and manage edge cases, such as invalid price inputs, unrealistic discount requests, or complex negotiation scenarios that might require special handling.

By implementing these components, the chatbot will be equipped to manage a variety of negotiation scenarios, interact intelligently with users, and provide a seamless negotiation experience. The goal is to ensure that the chatbot can handle user interactions effectively, generate appropriate responses, and facilitate successful negotiations.

Sentiment Analysis for Dynamic Price Adjustments

Sentiment Analysis for Dynamic Price Adjustments is a critical component of the negotiation chatbot that ensures the chatbot's responses are not only contextually relevant but also tailored to the user's emotional state. By integrating sentiment analysis into the negotiation process, the chatbot can adjust its pricing strategy dynamically based on the user's sentiments. Here's how it can be implemented:

1.Sentiment Analysis Overview:

- **Purpose:** Sentiment analysis helps understand the user's emotional tone and mood during the negotiation. This can include detecting emotions such as frustration, satisfaction, eagerness, or indifference.
- **Objective:** By analyzing sentiments, the chatbot can adjust its negotiation tactics, such as offering better deals to frustrated users or maintaining firm pricing with satisfied users.

2.Integration of Sentiment Analysis:

- **Library Utilization:** Use the NLTK (Natural Language Toolkit) library for sentiment analysis. NLTK provides tools and resources for processing and analyzing text, including sentiment analysis modules.
- **Sentiment Scoring:** Analyze the sentiment of user inputs using NLTK's sentiment analysis tools. This involves assigning sentiment scores or labels (e.g., positive, negative, neutral) to the text based on the emotional content.

3.Implementing Sentiment Analysis:

- **Text Preprocessing:** Before performing sentiment analysis, preprocess the user input to remove noise (e.g., special characters, stop words) and normalize the text (e.g., lowercasing).

- **Sentiment Classification:** Apply NLTK's sentiment analysis algorithms to classify the sentiment of the user's input. For instance, you might use pre-trained sentiment classifiers or build a custom model based on labeled training data.

4.Dynamic Price Adjustment Logic:

- **Adjustment Criteria:** Define criteria for adjusting prices based on sentiment scores. For example:

- **Positive Sentiment:** If the sentiment is positive, the chatbot might maintain or slightly adjust the initial offer, reflecting the user's satisfaction.

- **Neutral Sentiment:** For neutral sentiment, the chatbot might offer standard counteroffers or discounts based on the negotiation rules.

- **Negative Sentiment:** If the sentiment is negative, the chatbot might offer additional discounts or better terms to address the user's dissatisfaction and encourage continued negotiation.

- **Price Modification:** Implement logic to modify the proposed price or offer discounts dynamically based on the sentiment analysis results. This could involve adjusting the offer percentage or introducing special terms to improve user experience.

5.Feedback Loop:

- **Continuous Monitoring:** Continuously monitor and analyze user sentiment throughout the negotiation. This allows the chatbot to adapt its responses in real-time based on ongoing sentiment changes.
- **Response Adaptation:** Adjust the chatbot's negotiation strategy based on sentiment trends. For example, if a user's sentiment improves, the chatbot might scale back on aggressive discounting.

Evaluation and Optimization:

- **Testing:** Regularly test the sentiment analysis and dynamic adjustment logic to ensure it effectively improves user experience and negotiation outcomes.

- **Feedback Integration:** Gather feedback from users to fine-tune the sentiment analysis thresholds and adjustment criteria, ensuring the chatbot remains responsive and effective.

By incorporating sentiment analysis into the chatbot's negotiation logic, the system becomes more responsive to user emotions, leading to a more personalized and engaging negotiation experience. This approach helps in building a more effective and empathetic negotiation chatbot that can adapt to various emotional states and improve overall user satisfaction.

API Interaction with Gemini

API Interaction with Google Gemini involves integrating the chatbot with the Gemini AI model or a similar AI model to handle the conversational aspects of negotiation. This section covers how to set up and interact with the Gemini API to facilitate dynamic and intelligent responses during the negotiation process.

Introduction to Gemini API:

- **Purpose:** The Gemini API provides access to advanced natural language processing capabilities, enabling the chatbot to generate contextually relevant and conversational responses based on user inputs.

- **Functionality:** The API allows the chatbot to send user inputs to Gemini and receive responses that can include text-based counteroffers, negotiation suggestions, and other relevant interactions.

1.API Setup:

- **API Key:** Obtain an API key from the Gemini service provider. This key is required for authenticating requests and accessing the API endpoints.

- **Endpoint Information:** Familiarize yourself with the API endpoints provided by Gemini. These endpoints handle different types of interactions, such as sending user inputs and receiving responses.

2.Sending Requests to Gemini:

- **Request Structure:** Construct the HTTP requests to interact with the Gemini API. This typically involves sending a POST request with user input data and receiving a response with generated text.
- **Parameters:** Include necessary parameters in the request, such as user input text, context, and any specific instructions for generating responses.

3.Handling API Responses:

- **Response Parsing:** Parse the responses received from the Gemini API. This involves extracting the relevant information from the API's JSON response format.

- **Response Integration:** Integrate the responses into the chatbot's logic to present them to the user. This can include displaying text-based counteroffers or suggestions for negotiation.

4.Error Handling:

- **API Errors:** Implement error handling to manage cases where the API request fails or returns an error response. This can include retries, logging errors, and notifying users of issues.
- **Invalid Responses:** Handle scenarios where the API returns unexpected or malformed responses by validating the response data and providing fallback messages.

5.Security Considerations:

API Key Security: Keep your API key secure by storing it in environment variables or secure storage, and avoid hardcoding it directly in the codebase.

Rate Limits: Be aware of any rate limits imposed by the Gemini API and ensure that your application handles limits gracefully to avoid service disruptions.

6.Optimization and Testing:

- **Performance Testing:** Test the performance and responsiveness of API interactions to ensure that the chatbot provides timely and accurate responses.

- **User Feedback:** Collect feedback from users to refine the interaction model and improve the quality of the responses generated by Gemini.

By integrating the Gemini API into your chatbot, you enable it to leverage advanced natural language processing capabilities, making the negotiation process more dynamic and engaging. This setup allows the chatbot to handle complex conversational scenarios and provide contextually relevant responses that enhance the overall user experience.

Streamlit for User Interface

Streamlit for User Interface

Streamlit is a popular open-source framework designed to create interactive web applications quickly and easily, particularly for data science and machine learning projects. For the negotiation chatbot project, Streamlit is used to build an intuitive and user-friendly interface that allows users to interact with the chatbot seamlessly.

Introduction to Streamlit:

- **Purpose:** Streamlit provides a straightforward way to develop web applications without the need for extensive front-end coding. It allows developers to focus on functionality and user experience, leveraging Python code to build interactive applications.
- **Features:** Streamlit offers built-in components for displaying text, images, charts, and interactive widgets, making it ideal for creating a dynamic user interface for the chatbot.

Setting Up Streamlit:

Installation: Install Streamlit using pip if it's not already installed. Run the following command:

[pip install streamlit]

Basic Setup: Create a Python script **(app.py)** that will serve as the main entry point for the Streamlit application.

Building the User Interface:

- **Layout and Design:** Use Streamlit components to design the layout of the chatbot interface. This typically includes text input boxes, buttons, and output areas to display the chatbot's responses.

Interactive Elements:

- **Text Input:** Use st.text_input() to allow users to input their negotiation offers or messages.
- **Buttons:** Use st.button() to trigger actions such as sending the user's input to the chatbot and receiving a response.
- **Output Display:** Use st.write() to display the chatbot's responses dynamically.

Customizing the Interface:

- **Styling:** Customize the appearance of the Streamlit app using Markdown and HTML elements if needed. Streamlit supports basic styling through Markdown.

- **Interactive Widgets:** Add more interactive widgets like sliders, checkboxes, or dropdown menus if your chatbot requires additional user inputs or options.

Testing and Iteration:

- **User Testing:** Test the user interface with real users to gather feedback and make improvements based on usability and user experience.

- **Performance:** Monitor the performance of the Streamlit application to ensure that it remains responsive and efficient under different usage scenarios.

Deployment:

Local Deployment: Run the Streamlit application locally using:

[streamlit run app.py]

By using Streamlit for the user interface, we create an interactive and engaging experience for users to interact with the negotiation chatbot. The simplicity of Streamlit allows you to focus on building a functional and effective chatbot without extensive front-end development.

Results and Conclusion

Results

1. System Performance and Accuracy:

Chatbot Accuracy: The negotiation chatbot demonstrated effective performance in handling price negotiations. Based on interactions, the chatbot accurately provided counteroffers and adjusted offers based on the pricing logic implemented. The AI model, whether Gemini or ChatGPT, was able to maintain context and provide relevant responses to user inputs.

Sentiment Analysis: The sentiment analysis, powered by the NLTK library, played a crucial role in adjusting responses based on user sentiment. The chatbot successfully adapted its negotiation strategy according to the perceived sentiment, whether it was positive, neutral, or negative.

2. User Interaction and Experience:

User Feedback: Users found the chatbot's responses to be relevant and realistic, simulating a genuine negotiation process. The interactive UI built with Streamlit allowed for seamless user interaction, making the negotiation process intuitive and engaging.

Negotiation Cycles: The chatbot effectively handled multiple negotiation cycles, providing counteroffers and discounts as needed. This functionality was well-received by users who appreciated the dynamic nature of the negotiation process.

3. Performance Metrics:

Response Time: The application maintained a quick response time, ensuring a smooth and efficient user experience. The interaction with the Gemini or ChatGPT API was prompt, allowing users to receive real-time responses.

Accuracy of Sentiment Adjustments: The sentiment analysis was accurate in adjusting the chatbot's responses, enhancing the negotiation process. The accuracy of sentiment detection and the subsequent adjustments were validated through testing and user feedback.

Conclusion

The negotiation chatbot project successfully developed an AI-driven tool capable of simulating price negotiations between customers and suppliers. Using a combination of Streamlit for the user interface, NLTK for sentiment analysis and a pre-trained AI model Google Gemini AI for conversational responses, the chatbot effectively handled negotiations and adjusted responses based on user sentiment. The system demonstrated accurate performance, quick response times, and a positive user experience. Future improvements could include exploring advanced AI models, adding new features, and scaling the application for broader use. Overall, the project showcases the potential of AI in enhancing interactive business processes.