Integration of the Gemini AI Model for a Negotiation Chatbot

Overview:

This documentation explains how I integrated the Gemini AI model and TextBlob for building a Negotiation Chatbot that handles price-based negotiations with users. The chatbot utilizes a pre-trained Gemini AI model for generating responses during the negotiation and TextBlob for sentiment analysis of the user's input.

Requirements:

- Google Generative AI Python SDK
- TextBlob (for sentiment analysis)
- Python 3.x

Gemini AI:

Gemini AI is a powerful generative model that enhances the chatbot's ability to engage in dynamic and natural conversations. It excels in generating human-like responses, making it ideal for negotiating counteroffers in real-time. With its ability to adapt to specific prompts and maintain context throughout interactions, Gemini AI ensures fluid, contextually relevant conversations, improving the overall user experience.

TextBlob:

TextBlob is a lightweight and user-friendly tool for performing sentiment analysis. In the chatbot, it quickly analyzes the emotional tone of the user's input, allowing for more empathetic and tailored responses. Its simplicity and efficiency make it an excellent choice for real-time applications, where adjusting responses based on user sentiment is key to improving interaction quality.

API Configuration:

```python
import google.generativeai as genai
from textblob import TextBlob

# Configure the Gemini AI API key
api_key = "API_KEY"
genai.configure(api_key=api_key)

# Define price limits for negotiation
min_price = 1000
max_price = 4000
```

The script starts by configuring the Gemini AI API using the provided API key. This allows interaction with the Gemini AI models for response generation. Price limits are set within which the chatbot will negotiate. If the user offers a price outside this range, a counteroffer will be generated.

Gemini AI Model Integration:

Function to Generate AI Responses:

```python
def get_gemini_response(prompt):
    model = genai.GenerativeModel("gemini-pro")
    chat = model.start_chat(history=[])

    response = chat.send_message(prompt)

    # Combine the response into a single string
    generated_text = ''.join([chunk.text for chunk in response])
    return generated_text
```

This function uses the Gemini AI model to generate responses. We initiate a chat session with the AI, provide a prompt for response generation and return the model's generated response. The generated text is combined into a single string for easy interpretation.

Main Negotiation Function:

```python
def negotiate(user_input):
    # Analyze sentiment of the user input
    sentiment = analyze_sentiment(user_input)

    # Extract price from user input
    user_price = extract_price(user_input)

    # Negotiate based on the price
    if user_price is not None:
        if is_offer_acceptable(user_price):
            if sentiment > 0:
                # Positive sentiment: Offer a better deal
                generated_text = f"That's a great offer! Here's a 10% discount, making it ${user_price * 0.9:.2f}"
            else:
                # Neutral/Negative sentiment: Accept offer
                generated_text = f"I'm happy to accept your offer of ${user_price}. We have a deal!"
            return generated_text, True  # Return with a flag to end the loop
        else:
            # Generate a counteroffer using the Gemini model
            prompt = f"The user offered {user_price}. The acceptable range is between {min_price} and {max_price}
            generated_text = get_gemini_response(prompt)
            return generated_text, False  # Continue the negotiation
    else:
        return "Sorry, I couldn't understand the price you mentioned. Can you please clarify?", False
```

The negotiate function handles the core logic of the chatbot. Uses `TextBlob` to assess whether the user's input is positive, negative or neutral. Extracts numerical values from the user input, assuming it contains a price. If the extracted price is within the acceptable range, a response is generated based on the sentiment. If the price is outside the acceptable range, a counteroffer is generated using Gemini AI. The function returns a response and a flag indicating whether the negotiation is complete.

Sentiment Analysis:

```python
def analyze_sentiment(text):
    blob = TextBlob(text)
    sentiment = blob.sentiment.polarity
    return sentiment
```

This function analyzes the sentiment of the user input using TextBlob. It returns a sentiment polarity score between -1 (negative) to +1 (positive).

Price Extraction Logic:

```
def extract_price(text):
    words = text.split()
    for word in words:
        try:
            price = int(word)
            return price
        except ValueError:
            continue
    return None

```

This function parses the user's input to extract the first integer (assumed to be the offered price). If no valid price is found, it returns `None`.

Price Acceptance Check:

```
def is_offer_acceptable(user_price):
    return min_price <= user_price <= max_price
```

This helper function checks whether the user's offer falls within the defined price range (min_price to max_price).

Main Loop:

```
if __name__ == "__main__":
    while True:
        user_input = input("Enter your offer: ")
        response, negotiation_done = negotiate(user_input)
        print(response)

        if negotiation_done:
            print("Negotiation complete. Thank you!")
            break  # Exit the loop after a deal is made
```

The main loop continuously prompts the user for their input. After each negotiation, the chatbot prints its response and checks if the negotiation is complete. The loop breaks when a deal is reached.

Summary:

This negotiation chatbot project successfully integrates Gemini AI for generating human-like, context-aware responses and TextBlob for real-time sentiment analysis, creating a dynamic and adaptive interaction experience. By leveraging these tools, the chatbot can handle price-based negotiations with users, adjusting responses based on sentiment and price thresholds to ensure smooth and intelligent conversations. The implementation demonstrates how advanced AI models and natural language processing can be used together to build smart, responsive systems, enhancing user experience in real-world applications.