Integration Documentation for Negotiation Chatbot API

Atul sah
atulsah9211@gmail.com
7905740262
GitHub: Atulsah17

Overview:

This document provides a detailed explanation of how the generative AI model and sentiment analysis are integrated into the FastAPI-based negotiation chatbot. The chatbot handles negotiations for a Bluetooth speaker, providing dynamic responses based on user input and sentiment.

Components

1. **FastAPI**: A modern web framework for building APIs with Python.
2. **Google Gemini**: Used for generating AI responses.
3. **TextBlob**: Used for sentiment analysis of user inputs.
4. **Logging**: Used for monitoring and debugging.

Integration Details

1. FastAPI Setup

The chatbot API is built using FastAPI, which is a high-performance web framework for building APIs. The FastAPI application is initialized in the `chatbot.py` file.

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
```

```
import google.generativeai as genai
import os
from dotenv import load_dotenv
from textblob import TextBlob
import logging

# Load environment variables
load_dotenv()

# Initialize FastAPI app
app = FastAPI()
```

2. Google Gemini Configuration

Google Gemini is configured for generating AI responses. The API key is loaded from an environment variable using the `dotenv` library.

```
# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Configure Gemini API
try:
    genai.configure(api_key=os.getenv("GEMINI_API_KEY"))
    logger.info("Gemini API configured successfully.")
except Exception as e:
    logger.error(f"Failed to configure Gemini API: {e}")
    raise HTTPException(status_code=500, detail="Failed to configure Gemini
API.")
```

3. Sentiment Analysis with TextBlob

Sentiment analysis is performed using the TextBlob library to determine the sentiment polarity of user input. The sentiment score helps decide the response strategy.

```
def analyze_sentiment(user_input):
    try:
        analysis = TextBlob(user_input)
        return analysis.sentiment.polarity
    except Exception as e:
        logger.error(f"Sentiment analysis failed: {e}")
        return 0.0  # Default to neutral if analysis fails
```

5. Generating AI Responses

For neutral or unclear sentiments, the chatbot uses Google Gemini for response generation. The response is cleaned to remove unnecessary characters.

```
def clean_response(response_text):
    return response_text.replace("\n", " ").strip()

try:
    model = genai.GenerativeModel(model_name="gemini-1.5-flash",
generation_config=generation_config)
    chat = model.start_chat(history=conversation_history)
    response = chat.send_message(user_input.message)
    bot_response = clean_response(response.text)
except Exception as e:
    logger.error(f"Error during AI model response generation: {e}")
    raise HTTPException(status_code=500, detail=f"Error generating response:
{e}")
```

6. API Endpoints

- **Health Check (GET /)**: Confirms that the API is running.
- **Negotiate (POST /negotiate)**: Processes negotiation messages, analyzes sentiment, and generates responses.

Example API Usage

Health Check

```
curl -X 'GET' 'http://127.0.0.1:8000/'
```

Response:

```
{
  "message": "Welcome to the Bluetooth speaker negotiation bot!"
}
```

Negotiate:

```
curl -X 'POST' \
  'http://127.0.0.1:8000/negotiate' \
  -H 'Content-Type: application/json' \
  -d '{"message": "This seems overpriced, I don’t think it’s worth more than $70."}'
```

Response:

```
{
  "response": "Your offer of $70 is quite low. The minimum acceptable price is $100. Can we try to meet closer to that?"
}
```

Conclusion

This documentation outlines the integration of the generative AI model and sentiment analysis into the FastAPI-based negotiation chatbot. By combining these technologies, the chatbot effectively engages in negotiations, handles basic product inquiries, and responds dynamically based on user sentiment.