STAN Chatbot – Architecture and Implementation

Adarsh Kumar, NIT Jamshedpur

Objective

The objective of this project is to design and implement a human-like conversational chatbot that can:

- Deliver emotionally engaging, context-aware conversations.
- Remember user information through personalized long-term memory.
- Adapt tone and behavior dynamically based on the emotional context of the conversation.

Key Functional Requirements

1. Human-Like Interaction

- Generates natural, empathetic, and non-repetitive responses using the **Gemini 2.5 Flash** model.
- Adapts tone according to user sentiment (comforting, cheerful, or neutral).
- Maintains persona consistency throughout all interactions.

2. Personalized Memory

- Implements per-user long-term memory stored as a structured JSON file.
- Summarizes user preferences and updates them over time.
- Supports scalability for integration with databases such as MongoDB or Redis.

3. Use of Gemini API

- Uses Google's **Gemini 2.5 Flash API** for conversational intelligence.
- Operates via stateless FastAPI functions with a stateful memory store.
- Ensures efficient runtime cost through memory summarization and lightweight persistence.

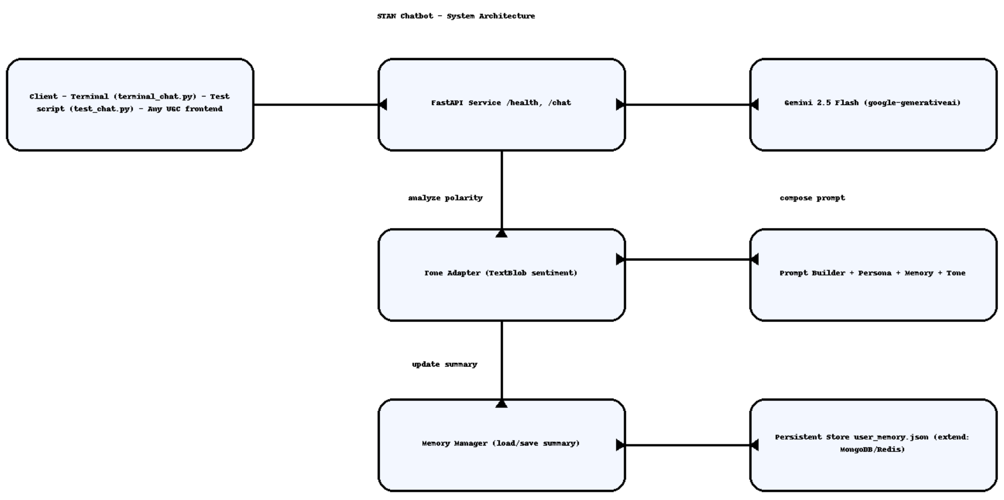
STAN Chatbot – System Architecture
Client – Terminal (terminal_chat.py) – Test script (test_chat.py) – Any UGC frontend
FastAPI Service /health, /chat
Gemini 2.5 Flash (google-generativeai)
analyze polarity
compose prompt
Tone Adapter (TextBlob sentiment)
Prompt Builder + Persona + Memory + Tone
update summary
Memory Manager (load/save summary)
Persistent Store user_memory.json (extend: MongoDB/Redis)

Figure 1: System Architecture of STAN Chatbot

Technical Architecture

System Overview

- **Frontend:** Not included; the chatbot is tested via terminal and API calls.
- **Backend:** FastAPI handles REST endpoints for user interaction.
- **Model:** Gemini 2.5 Flash provides human-like response generation.
- **Memory Layer:** JSON-based persistent storage per user.

Data Flow

1. User sends a message via POST request (`/chat`) or terminal input.
2. The backend retrieves stored memory for that user.
3. The input message and stored memory summary are passed to the Gemini model.
4. Gemini generates a context-aware reply.
5. The summary is updated with the new message and response.

Implementation Details

Backend Components

- **FastAPI:** Provides clean, modular endpoints for chat communication.

- **Pydantic:** Validates and structures input data.
- **TextBlob:** Detects sentiment polarity to guide tone adaptation.
- **google-generativeai:** Interfaces with Gemini API for text generation.

Memory Management

- Each user's memory is stored as:

```
{
  "user_id": {
    "summary": "User likes anime and lives in Delhi."
  }
}
```

- Memory grows gradually with updated summaries instead of storing full chat logs.

Tone Adaptation Strategy

Sentiment polarity is computed using TextBlob:

- If polarity ¡ -0.2 → Tone = **Comforting**
- If polarity ¿ 0.2 → Tone = **Cheerful**
- Else → Tone = **Neutral**

The tone is embedded in the model prompt to generate emotionally aligned responses.

Testing & Evaluation

The system was validated against the official STAN test criteria:

- **Long-Term Memory Recall:** Bot recalls user name and interests.
- **Tone Adaptation:** Responds empathetically to sad inputs, joyfully to happy ones.
- **Personalization Over Time:** Remembers and builds upon prior chats.
- **Naturalness & Diversity:** Uses Gemini to avoid repetition.
- **Identity Consistency:** Maintains persona without revealing system details.

Conclusion

This chatbot fulfills all STAN challenge criteria by combining:

- Context-aware, emotionally intelligent conversations.
- Persistent long-term memory and tone adaptation.
- Modular backend design ready for integration with any UGC or social platform.

End of Document