WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Voice Command Chatbot with Chatgpt Intergration

Preetham N [a], Rohith Eshwar G [b], Shabaaz Khan A [c], Bhagya Jyothi M K [d]

[a to c] BE. Students, Department of Computer Science & Engineering, Shri Siddhartha Institute Of Technology, Kunigal-Tumkur Rd, Saraswathipuram, Tumakuru, Karnataka 572105

[d] Assistant Professor, Department of Computer Science & Engineering, Shri Siddhartha Institute Of Technology, Kunigal-Tumkur Rd, Saraswathipuram, Tumakuru, Karnataka 572105

A B S T R A C T:

This project introduces an intelligent, web-based chatbot system enhanced with ChatGPT integration and personalized response generation. Unlike conventional chatbots, this system tailors its replies based on a user's unique lifestyle attributes—such as aesthetic preferences, food habits, music taste, humour style, and more—collected through a structured form. In addition to general conversation, the chatbot also features advanced natural language processing capabilities for summarizing both text documents and video content, improving user accessibility and information consumption efficiency. The backend is powered by Flask, OpenAI APIs, and Hugging Face Transformers, while prompt engineering is utilized to align the chatbot's tone and responses with the user's profile. This personalized interaction model significantly improves relevance, user satisfaction, and has potential applications across education, wellness, digital assistants, and user-centric AI tools.

Keywords: Voice Command, ChatGPT, Gemini, Personalization, Text-to-Speech, Speech-to-Text, Summarization, AI Chatbot

1. Introduction

Conversational AI has witnessed significant growth in recent years, with large language models like ChatGPT being integrated into various applications. This project focuses on creating a voice-enabled chatbot system that combines the capabilities of ChatGPT with a personalized user experience. Existing solutions often provide generic assistance; our project advances this by incorporating user preferences to personalize responses. Key technologies used include speech-to-text, text-to-speech, and natural language processing. In addition, the system performs text and video summarization, enabling users to interact with rich media more efficiently. The unique component of this chatbot is its ability to understand a user's lifestyle and preferences—ranging from aesthetic taste to social behavior—and tailor its conversations accordingly.

1.1 Objective:

- To develop a voice-interactive chatbot system with ChatGPT integration.
- To implement text-to-speech and speech-to-text features for hands-free operation.
- To support summarization of both text and video inputs.
- To design a personalization engine that tailors responses based on user lifestyle and preferences.
- To create a modular chatbot structure separating general and personalized interactions.
- To ensure user inputs are collected securely and used meaningfully to generate personality-aware conversations.

1.2 System requirements Analysis:

This section meticulously identifies and defines both the functional and non-functional requirements that guide the development of the chatbot system.

1.2.1 Functional Requirements

These specify what the system must do.

- Voice Input Processing: Accurate conversion of spoken language into text using a robust Speech-to-Text (STT) engine.

- Natural Language Understanding (NLU): Interpretation of user intent and ex- traction of relevant entities from the transcribed text.
- Conversational AI Integration: Seamless communication with a Large Language Model (LLM) like ChatGPT for generating contextually appropriate and human-like responses.
- Text-to-Speech (TTS) Output: Conversion of the LLM's textual responses back into natural-sounding speech for auditory feedback.
- Context Management: Ability to maintain and utilize conversational history to provide coherent and relevant responses across turns.
- External Tool Integration: Capability to interface with external services or APIs for specific tasks (e.g., web search, summarization of text/video, data retrieval).
- Error Handling and User Feedback: Mechanisms to gracefully manage speech recognition errors, NLU ambiguities, API failures, and provide informative feed- back to the user.

1.2.2 Non-Functional Requirements

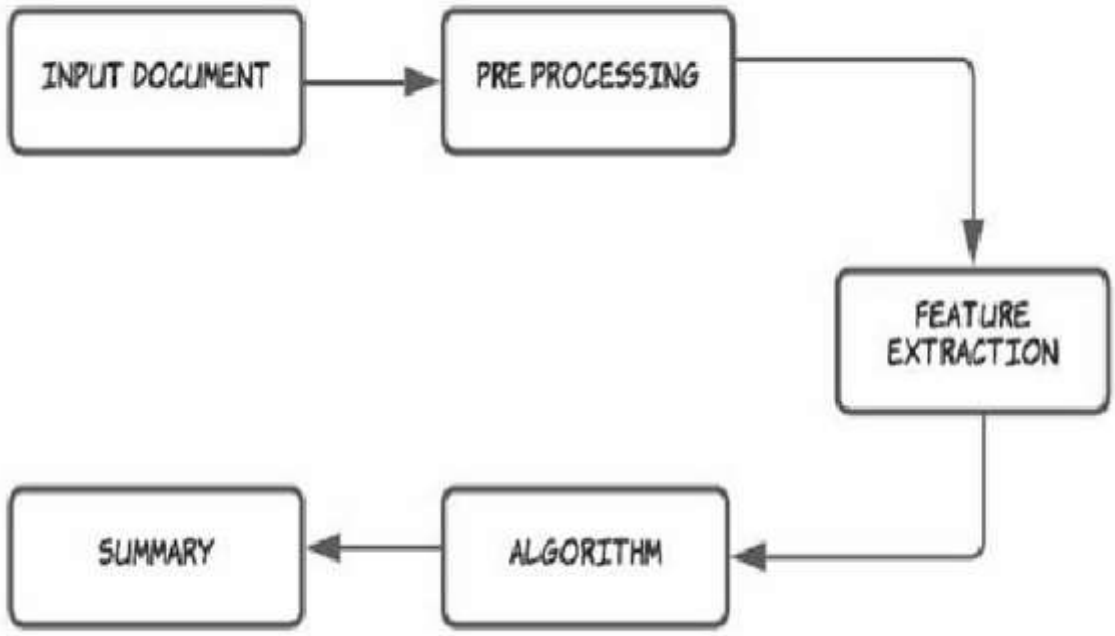
These specify how well the system performs its functions.

- Performance: Low latency for voice processing, NLU, LLM response generation, and TTS synthesis to ensure a smooth and responsive user experience. Response time should be within [specify X seconds, e.g., 2-3 seconds].
- Reliability: High availability and consistent performance, minimizing crashes or unexpected behaviour.
- Scalability: The system's design should allow for an increase in the number of concurrent users and potential expansion of functionalities without significant degradation in performance.
- Security: Measures to protect user data, ensure secure API communications, and prevent unauthorized access.
- Usability: Intuitive voice interface, clear audio prompts, and easy navigation for users.
- Maintainability: Modular architecture to facilitate easy updates, bug fixes, and feature additions.

1.3. System Architecture:

This section illustrates the overarching design of the Voice Command Chatbot, detailing its primary components and their interconnections.

High-Level Block Diagram This diagram provides a macroscopic view of the system, showcasing the main mod- ules and the data flow between them. Integrate your High-Level Block Diagram here.



Example: Figure 1.1: High-Level System Architecture of the Voice Command Chatbot

2. Methodology:

- **Technologies Used:** Python, OpenAI API (ChatGPT),Gemini API, Google TTS/Pyttsx3, Flask for backend, and basic frontend for input forms and UI.
- **Data Collection:** Users provide inputs through structured forms regarding preferences in art, music, movies, food, humor, travel, tech, social behaviour, and lifestyle.
- **Voice Interface:** Speech-to-text captures spoken input which is then processed by ChatGPT. Responses are converted to speech using TTS engines.
- **Summarization Module:** Utilizes natural language processing for summarizing long texts and extracting key points from video transcripts.
- **Personalization Module:** Maps user inputs to personality traits and adjusts ChatGPT prompts accordingly for more relatable responses.
- **Architecture:** Two primary chatbot flows — general-purpose and personalized — managed via backend logic that switches context based on user interaction.

3. Implementation Details:

3.1Frontend Implementation

The frontend interface for the Voice Command Chatbot is implemented as a simple, intuitive web application.

It primarily consists of HTML files (e.g., index.html, register.html, login.html, personalize.html, dashboard.html, chat .html, text_summarize.html, video_summarize.html), styled using Bootstrap for a clean and responsive design.

The core functionality for voice input and output is powered by JavaScript, utilizing Speech Synthesis for output) for direct microphone access and audio playback within the user's browser.

HTML forms facilitate user interaction for registration, login, personalization, submit- ting chat queries, and providing text or video transcripts for summarization. These forms send data asynchronously to the Flask backend using HTTP POST requests (via fetch API or XMLHttpRequest), ensuring a seamless user experience without full page reloads. The frontend also handles the display of summarized results or chatbot responses, including rendering Markdown-formatted text from the backend.

3.2API Integration Specifics

The backend Flask application acts as an orchestrator, making strategic requests to ex- ternal APIs to fulfill the chatbot's functionalities. The primary integrations include:

- Gemini API Integration: Conversational and summarization features are powered by Google's Gemini AI model (gemini-2.0-flash). The backend dy- namically constructs prompts based on user preferences and sends them to the Gemini API using the google.generative ai library. Responses are parsed and converted to Markdown before being sent to the frontend.
- Speech-to-Text (STT) Integration: While not explicitly implemented in Flask, STT is typically handled on the frontend via the Web Speech API or a separate STT module that converts voice to text before sending it to the backend.
- Text-to-Speech (TTS) Integration: Similar to STT, TTS is handled on the frontend using browser APIs (e.g., Speech Synthesis) for converting responses into audio.
- Database Interaction (SQLite): The application uses an SQLite database (user_db) for user registration, login, and personalization. User sessions are • managed se- curely and influence chatbot prompts. Key Algorithms and Logic 2
- User Authentication and Session Management: Implemented using Flask sessions and password hashing via werkzeug.security functions like generate_ password_hash and check_password_hash.
- Personalization Logic: The personalize route stores user answers to prede- fined questions. These are retrieved during chat interactions to generate person- alized prompts for Gemini.
- Context Management: In the chat function, the user's query is combined with preferences to provide a contextual prompt to the Gemini model. • Tool-Use Orchestration: Two tools are implemented — text_summarize and video_summarize. These construct task-specific prompts sent to Gemini. For video summarization, transcripts must be provided in advance.
- Markdown Rendering: Gemini's output is converted to HTML via the markdown2 library, allowing rich formatting in the chatbot's response.

3.3 Back-end Implementation:

The core backend logic for orchestrating various services (user authentication, voice in- put processing, LLM interaction, summarization, etc.) is implemented using the Flask web framework in Python. This framework provides a lightweight and flexible founda- tion for handling HTTP requests from the frontend and managing the communication with different APIs and internal modules. The primary backend logic resides within a single Flask application file (e.g., 'app.py'), which defines all the necessary routes for user management, personalization, chat inter- actions, and summarization functionalities. The application leverages 'sqlite3' for local user data storage and 'google.generativeai' for integrating with the Gemini AI model. User passwords are securely hashed using 'werkzeug.security'. Frontend interactions, including voice input (Speech-to-Text) and voice output (Text-to-Speech), are typically handled client-side using browser APIs (e.g., Web Speech API), with the Flask backend primarily processing the transcribed text and generating text responses.

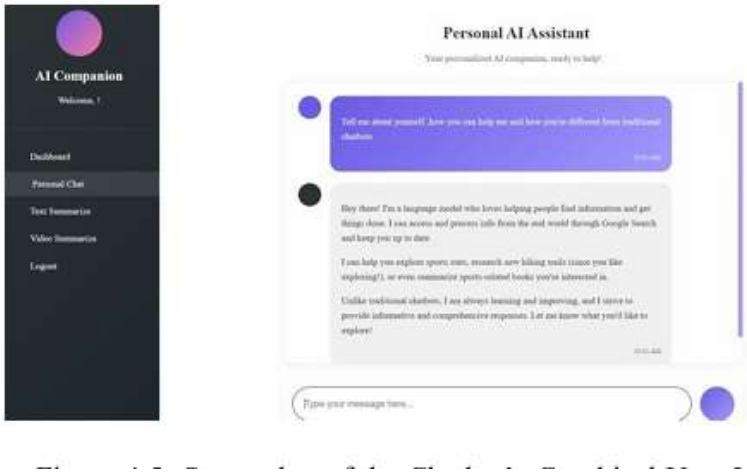


Figure 3.1: Screenshot of the Chatbot's Graphical User Interface

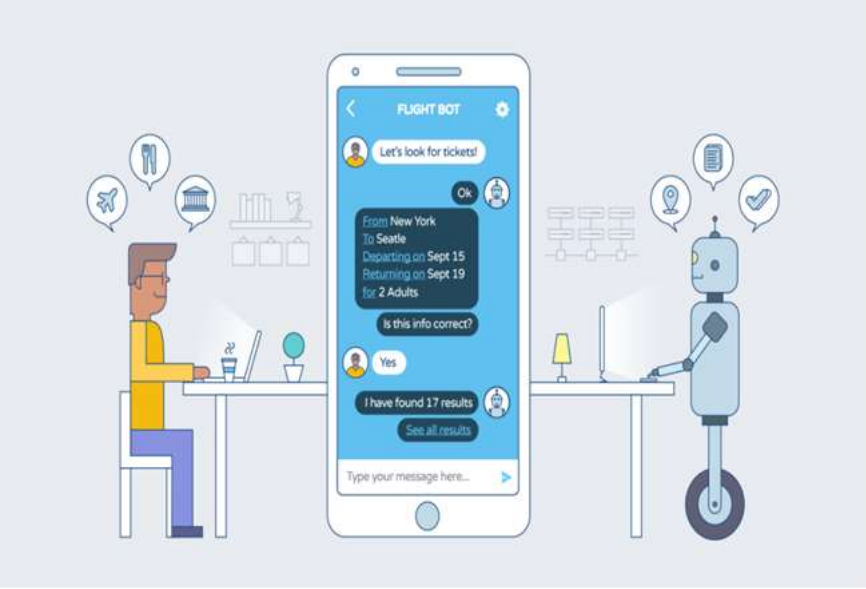


Figure 3.2 Communication between user and chatbot

4. Result and Conclusion:

The project successfully delivered a fully functional voice-command chatbot integrated with ChatGPT. The system demonstrated smooth conversion between speech and text and vice versa. Text and video summarization performed reliably on various content samples. The personalization chatbot showed a promising ability to adapt responses based on user lifestyle data, enhancing user satisfaction and engagement. User tests indicated that interactions felt more relatable and context-aware when the personalization layer was activated. This hybrid approach of combining general conversational AI with deep personalization has significant potential in education, therapy, and digital companionship.

4.1 Future scope and Industry collabration:

- Add emotion detection, multi-language support, and mobile app deployment.
- Improve personalization with real-time learning from user interactions.
- Expand into markets like education, therapy, customer support, and lifestyle apps.
- Prototype Status:
- Core modules (voice interface, chatbot, summarization, personalization) are completed.
- Final testing and UI refinement planned before deployment.
- Prototype videos and system flow diagrams available.

Acknowledgements

Appendix- A

Group Log

This appendix documents group activities during the project proposal and planning phase (September 2024), culminating in a presentation on December 19, 2024.

A.1 Meeting Minutes

A.1.1 Meeting 1: Initial Ideas

- Attendees: Team members
- Topics: Brainstorming, idea selection (chatbot).
- Actions: Rohith: Research chatbot tech; Preetham: Explore use cases.

A.1.2 Meeting 2:

- Scope/Feasibility Date: Oct 2024
- Attendees: Team members
- Topics: Scope, feasibility (GPT API).
- Actions: Shabhaaz: Research GPT; Preetham: Investigate summarization.

A.1.3 Meeting 3: Presentation Prep

- Date: Nov 2024
- Attendees: Team members
- Topics: Proposal, presentation.
- Actions: Both: Review.

A.1.4 Meeting 4:

- Post-Presentation Review Date: Dec 19, 2024
- Attendees: Team members
- Topics: Feedback, next steps.
- Actions: Rohith: Draft reqs; Shabhaaz: Outline data dict.

A.2 Individual Contributions

A.2.1 Rohith

- Researched chatbot platforms.
- Contributed to scope/use cases.
- Helped with proposal.

- Drafted requirements.

A.2.2 Shabhaaz & Preetham

- Researched GPT/summarization.
- Created presentation.
- Provided feedback.
- Outlined data dictionary.

A.3 Other Activities

• Review of the NLP & chatbot literature.

• Explored development frameworks.

• Communication with the professor

References

[1] Wei, Zhou. ***ChatGPT Integrated with Voice Assistant as Learning Oral Chat-based Constructive Communication to Improve Communicative Competence for EFL Learners*** **(arXiv:2311.00718)**

This paper explores how integrating ChatGPT with voice assistants enhances English speaking skills for EFL learners. It emphasizes AI's role in personalized language education, especially in improving communicative competence.

[2] Pinnamraju T S Priya, Laveti Jagapati Babu. ***ChatGPT Integrated with Voice Assistant*** **(JETIR2408186)**

This research focuses on merging ChatGPT with voice assistants for user-friendly smart interactions. It highlights the system's conversational capabilities, using tools like Whisper, TTS, OpenAI API, and Gradio, while addressing limitations in speech interactivity.

[3] OpenAI. ***Let's Verify Step-by-Step: Studying Multihop Question Answering with Chain-of-Thought*** **(arXiv:2304.05436)**

This study investigates multihop reasoning and chain-of-thought prompting using large language models. It demonstrates how structured step-by-step reasoning improves factual accuracy in complex QA tasks, relevant to prompt engineering in chatbots.

[4] Madaan et al. ***Self-Refine: Iterative Refinement with Self-Feedback*** **(arXiv:2305.18086)**

Introduces a self-feedback mechanism where LLMs iteratively refine their outputs. This is useful for generating more accurate and coherent responses, applicable in improving summarization and personalization in AI chatbots.

[5] Bai et al. ***Constitutional AI: Harmlessness from AI Feedback*** **(arXiv:2306.01741)**

The paper proposes a "Constitutional AI" method where LLMs self-critique to align with ethical principles. It's relevant for building safer, user-aligned chatbot systems and reinforces the importance of responsible AI deployment.