iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** VIII **Month of publication:** August 2025

DOI: https://doi.org/10.22214/ijraset.2025.73852

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VIII Aug 2025- Available at www.ijraset.com

A Web-Based Chatbot with Advanced Voice Control and Document Interaction Using Flask and JavaScript

Ashwini Maidamshetti[1], Ravikanth K[2]
[1]*Student, School of Informatics, Department of MCA, Aurora Deemed University, Hyderabad*
[2]*Assistant Professor, School of Engineering, Department of CSE, Aurora Deemed University, Hyderabad*

Abstract: ***Chatbots in the last few years have been the subject of growing interest as smart conversational agents for human–computer interaction in various applications. Most current systems, however, have limitations like insufficient flexible voice control, limited document support, and minimal user-friendly capabilities. To overcome these drawbacks, this paper introduces a web-based chatbot system designed using Flask as the backend and JavaScript as the frontend. The system incorporates cutting-edge features like dark mode for user ease, text-to-speech with voice toggle control, selective text reading, microphone-based speech input, and document upload (PDF, DOCX, TXT) for content-based question answering. The backend utilizes the OpenRouter API to provide accurate responses, while the frontend offers an interactive and accessible interface. Experimental evaluation and user feedback emphasize that the suggested system provides greater usability, flexibility, and engagement over traditional chatbots. This research shows the possibility of integrating multi-modal capabilities with lightweight web technologies to enhance user experience in conversational AI systems.***

Keywords: Chatbot, Natural Language Processing, Flask, JavaScript, Text-to-Speech, Voice Control, Document Q&A, Web Application

I. INTRODUCTION

Chatbots are software applications programmed to mimic human dialog and deliver interactive answers to user questions. As the use of artificial intelligence (AI) and natural language processing (NLP) gains momentum, chatbots have found their way into various fields like education, healthcare, customer support, and entertainment. Chatbots make things more accessible, decrease workload, and offer instant help to users. In spite of their popularity, most chatbot systems that are currently available have many drawbacks. Traditional chatbots tend to offer text-only interaction without voice integration and document handling capabilities. Users demand more intuitive and free-flowing interactions, such as the capability for voice control, file uploading for context-dependent response, and using a user-friendly interface. To fill these gaps, this paper proposes the design and implementation of a web-based chatbot with multiple interactive functionalities. The chatbot is deployed on Flask (Python) as the backend and JavaScript as the frontend, ensuring effortless communication among user interface and server. The system incorporates the following advanced features:

•\tDark Mode to promote user comfort while using the system.
•\tToggle Control for Text-to-Speech to allow users to switch voice output on or off as desired.
•\tSelective Reading of Text where the user can select which chatbot responses should be read aloud.
•\tSpeech Input through Microphone for hands-free interaction.
•\tUpload Document (PDF, DOCX, TXT) allowing users to question based on file content.

The backend utilizes the OpenRouter API to produce relevant and precise responses, and the frontend provides seamless interaction with a clean UI. Experimental verification confirms that the introduced chatbot achieves better usability, personalization, and flexibility than the conventional systems.

The key contributions of this research are as follows:

•\tDesign of a multi-feature chatbot with the integration of voice, text, and document-based interactions.
•Integration of a user-friendly interface with accessibility features like dark mode and voice toggle.
•Exhibition of document-aware Q&A capability, facilitating meaningful retrieval of information from uploaded documents.
•Comparison of system usability with traditional chatbot models.

©IJRASET: All Rights are Reserved

This paper is structured as follows: Section II contains related work and literature review. Section III discusses the system methodology and architecture. Section IV addresses results and evaluation. Section V concludes the work and offers future scope.

II. LITERATURE REVIEW

The decade has seen tremendous growth in chatbot systems that have moved from rule-based conversational agents to AI-based, multimodal conversational platforms with voice, text, and document-based communication capabilities. The literature evidences both technological progress and the ongoing issues of human–computer conversational systems, particularly in educational and customer support settings.

Sharma and Gupta [1] created a rule-based chatbot for educational inquiries, prioritizing student support through digitization by means of predefined answers. The system was not very adaptable and context-aware, limiting user satisfaction.

Mulyadi et al. [2] suggested an NLP and speech recognition-based voice-enabled chatbot with increased accessibility for visually impaired users. Although useful, the lack of voice toggle functionality limited its usability for regular users.

Chowdhury and Das [3] provided a summary of chatbot applications in higher education, emphasizing the requirement of document-aware systems to enhance learning assistance. Their conclusion showed insufficient interaction with file-based content.

Johnson [4] discussed document Q&A chatbots based on transformer models, observing substantial improvements in precision in information retrieval. Unfortunately, these models had high computational loads, which rendered them less compatible with web-based lightweight applications.

Kumar [5] exemplified a Flask-based chatbot for academic helpdesks with ease of deployment and integration. The drawback was that it only handled text responses without more sophisticated accessibility options.

Lee [6] ventured into multimodal chatbot platforms with both text and voice and emphasized user interaction gains. However, the inability to exercise selective text-to-speech control limited personalization.

Mozilla Developer Network [7] outlines the functionality of current JavaScript APIs for speech recognition and text-to-speech, which form a basis for adding interactive and lightweight chatbot functionality to web platforms.

Together, these pieces of work form the building blocks for the envisioned Web-Based Chatbot, which uses Flask as the backend, JavaScript as the frontend, and OpenRouter API for smart response generation, overhauling limitations of previous implementations of chatbots by adding voice toggle, selective reading of text, document Q&A, and accessibility features like dark mode.

III.METHODOLOGY

A. Existing Methodology

Currently available chatbot systems in both academic and commercial areas are mostly aimed at offering text-based dialog with minimal functionality like answering FAQs, customer questions, or general counseling. Such systems tend to be implemented as web or mobile apps with minimal user interfaces and API or rule-based engine integration. Although they enhance ease of access over manual support, current chatbots indicate shortcomings in voice flexibility, document interaction, and personalization of user interfaces.

1) Rule-Based Chatbots: These operate by running predefined scripts or decision trees.

- Key Characteristics:
  - Pattern-matching or keyword-based responses.
  - Deployed as basic web widgets or messenger bots.
  - Good for static FAQs and customer support.
- Limitations:
  - No a flexibility for unseen queries.
  - Cannot process documents or contextual data.
  - Lacks personal features like voice control or selective reading.

2) Voice-Enabled Chatbots: These systems integrate speech recognition and text-to-speech for interaction.

- Key Characteristics:
  - Speech-to-text input for hands-free communication.
  - Text-to-speech output for accessibility.
  - Useful in healthcare and customer service.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

- Limitations:
  - No toggle mechanism for enabling/disabling voice.
  - Often limited to one-way speech output without user control.
  - Minimal customization of voice interaction.

3) Document-Aware Chatbots: Advanced chatbots process uploaded files or large documents.

- Key Characteristics:
  - Uses NLP/deep learning models for question answering.
  - Supports PDF/Word document parsing.
  - Applied in education and enterprise knowledge bases.
- Limitations:
  - High computational cost
  - Not integrated into lightweight web platforms.
  - Lacks user-friendly interface for real-time usage.

Summary of Existing Methodologies:
While existing systems provide basic text or voice support**,** they lack combined multi-modal interaction (voice + text + file upload) and usability features such as selective reading, dark mode, and role-based control over chatbot behavior.

B. Proposed Methodology
The proposed methodology introduces a Web-Based Chatbot that combines text, voice, and document interaction within a single lightweight platform. It ensures usability, accessibility, and real-time performance, making it suitable for both academic and enterprise applications.

1) System Architecture
Implemented using Flask (backend) and JavaScript (frontend) with OpenRouter API for NLP.

- Frontend: JavaScript with chat interface, dark mode, voice toggle, and selective reading.
- Backend: Flask handling API requests, document parsing, and chat history management.
- API Integration: OpenRouter API generates intelligent responses.
- File Handling: Extracts and processes text from PDF, DOCX, and TXT files.
- Authentication (optional for extended use): Role-based access can be integrated for multi-user deployments.

2) Functional Modules

- Text Interaction: Standard text input and chatbot response.
- Voice Interaction: Microphone-based speech input + Text-to-Speech with ON/OFF toggle.
- Selective Reading: Only highlighted chatbot responses are read aloud.
- Document Upload: Supports PDF/DOCX/TXT → user can query based on file content.
- Chat History: Options to save or clear chat.
- Dark Mode: Improves readability and user comfort.

3) Data Flow / Work Flow:

- User sends a query via text or microphone.
- Backend (Flask) processes input and forwards it to OpenRouter API.
- Response is generated and displayed in the chat interface.
- If TTS is enabled, the response is read aloud.
- If a file is uploaded, the chatbot extracts content and answers file-based queries.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

4) Advantages of Proposed Methodology:

- Synthesizes multi-modal features (text, speech, document Q&A).
- Light-weight and scalable using Flask + JavaScript.
- Improved user accessibility with dark mode and selective text reading.
- Supports real-time document-aware conversation.
- Extensible for academic and enterprise domains.

IV. SYSTEM DESIGN AND ARCHITECTURE

The proposed Web-Based Chatbot System is designed using a three-tier architecture consisting of frontend, backend, and intelligence layers. Each layer plays a crucial role in ensuring smooth communication, feature integration, and real-time performance.

A. System Architecture Overview

The architecture of the chatbot system is illustrated in Fig. 1 and consists of the following layers:

1) Frontend Layer (Client-Side)
- Developed using HTML, CSS, and JavaScript.
- Provides the user interface for sending queries, uploading documents, and controlling chatbot features.
- Implements:
  - Dark Mode Toggle
  - Chat Window for text responses
  - Voice ON/OFF Control
  - Selective Text Reading
  - File Upload Module (PDF/DOCX/TXT)

2) Backend Layer (Server-Side)
- Built with Flask (Python).
- Manages communication between frontend and external API.
- Functionalities include:
  - Handling text and voice queries.
  - Extracting content from uploaded documents.
  - Managing chat history and save/clear functions.
  - Forwarding requests to the intelligence layer.

3) Intelligence Layer (API Integration)
- Uses OpenRouter API for NLP-based response generation.
- Processes both direct user queries and document-based queries.
- Ensures that responses are accurate, relevant, and context-aware.

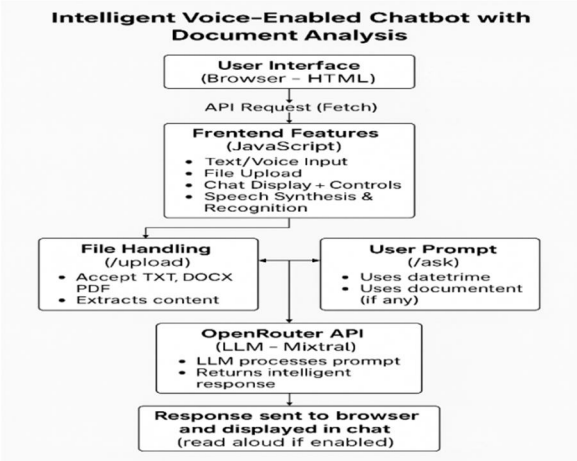


Fig. 1: System Architecture Diagram

B. System Components

1) User Interface (UI): Built with JavaScript for interactive experience.
2) Voice Module: Converts text to speech and speech to text with toggle and selective options.
3) Document Processing Module: Reads PDFs, DOCX, and TXT files, extracts content, and integrates with chatbot query.
4) Flask Backend: Handles all request–response cycles, ensures connectivity, and manages logic.
5) External API (OpenRouter): Provides intelligent and context-based chatbot responses.

C. Work Flow of Chatbot System

1) User Interaction – User enters text or speaks using the microphone.
2) Input Processing – The frontend sends input to the Flask backend.
3) API Query – Flask forwards the query to the OpenRouter API.
4) Response Generation – The API generates an intelligent response.
5) Output to User – The backend sends the response back to the frontend.
    - Displayed in chat interface.
    - Read aloud if TTS toggle is enabled.
    - If selective text is highlighted, only that part is spoken.
6) Document Interaction – If a document is uploaded, text is extracted and processed. User queries are answered specifically from file content.

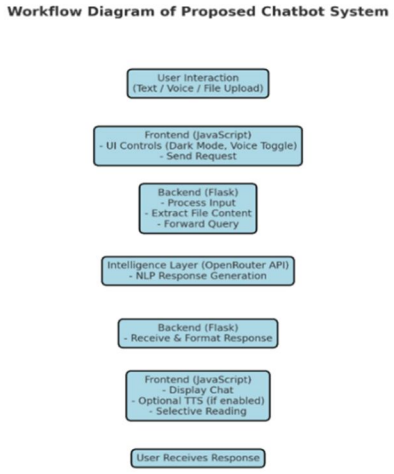


Fig. 2: Data Flow Diagram

V. IMPLEMENTATION

The Web-Based Chatbot was implemented using Flask (Python) for the backend and JavaScript (HTML, CSS) for the frontend to ensure lightweight deployment, usability, and real-time interaction. The system follows a modular architecture, where each feature (voice toggle, file upload, selective reading, chat management) is developed as an independent module.

A. Frontend Implementation

The frontend is developed using HTML, CSS, and JavaScript, styled for simplicity and responsiveness. Key frontend features include:

1) Dark Mode Toggle: Allows users to switch between light and dark themes for comfort.
2) Dynamic Chat Window: Displays user queries and chatbot responses in an interactive format.
3) Voice Control: Buttons to enable/disable text-to-speech (TTS) and a microphone button for speech-to-text (STT) input.
4) File Upload: Supports PDF, DOCX, and TXT formats for document-based queries.
5) Reusable Components: Chat bubbles, control buttons, and forms are designed as reusable UI elements.
6) API Integration: Uses fetch() for communication with backend Flask API.

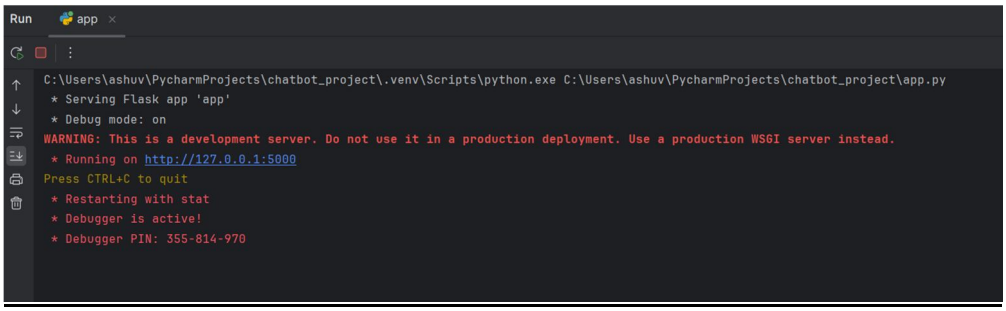


Fig. 3: Login Page

B. Backend Implementation

The backend is built with Flask (Python) and structured to handle modular functionality:

1) Controllers: Handle requests from frontend (text queries, file uploads, TTS toggle).

2) Routes:

•/generate → Processes user queries and returns responses from OpenRouter API.

• /upload → Extracts and stores text from PDF/DOCX/TXT files.

3) Document Parsing: Implemented with pypdf and python-docx to extract file content.

4) LLM Integration: OpenRouter API used to generate intelligent and context-aware responses.

5) Error Handling: Returns consistent JSON responses for invalid files or API errors.



Fig. 4: Open Router API Key

C. Database Implementation

Since the system is lightweight, no persistent database is used in the prototype. Instead, an in-memory dictionary temporarily stores uploaded document contents for Q&A.

1) Document Context Store: Maps uploaded file IDs to extracted text.

2) Chat History: Maintained temporarily for the user session, with options to save or clear.

3) Scalability: In extended deployments, a database such as MongoDB/PostgreSQL can be integrated for permanent storage of chat history and file data.

D. Key Modules Implemented

1) User Interaction Module

- Users can type or speak queries.
- Microphone input uses Web Speech API for speech-to-text conversion.

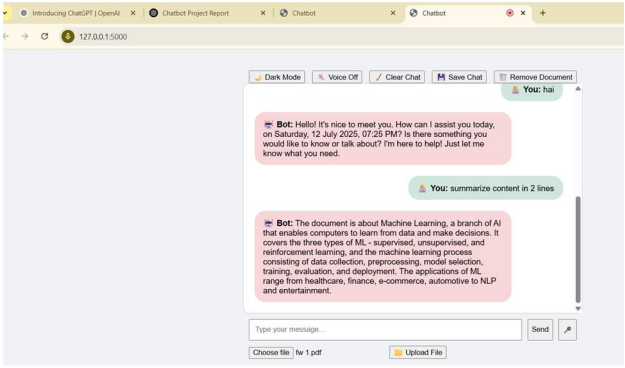


Fig. 5: Speech Input Using Microphone

2) Voice Control Module

- Text-to-Speech responses with ON/OFF toggle.
- Selective Reading: Users can double-click or right-click a response to hear only that text.

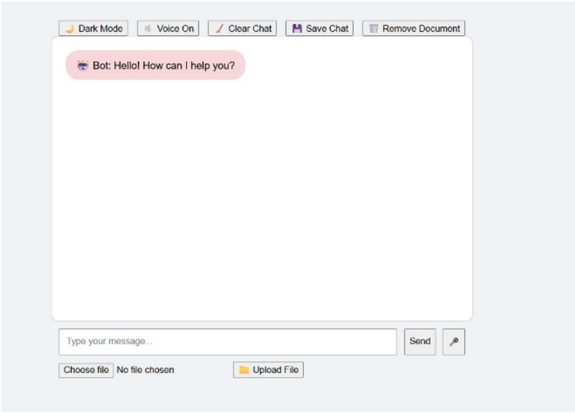


Fig. 6: Voice Toggle and Selective Reading in Action

3) Document Q & A Module

- Supports PDF, DOCX, and TXT uploads.
- Extracted text stored temporarily; users can ask queries directly from document content.

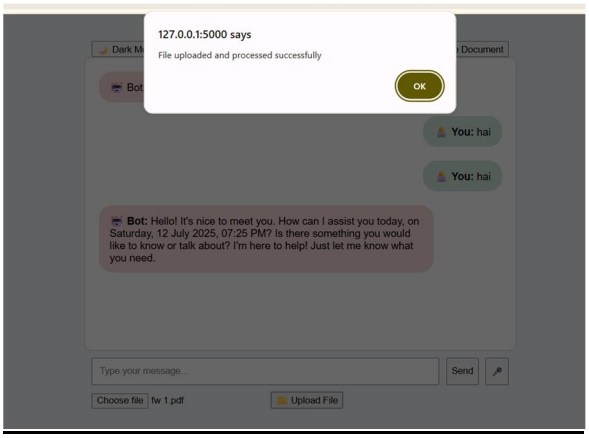


Fig. 7: Document Upload and Query Work Flow

4) Chat Management Module

- Options to Save Chat (download as file) and Clear Chat.
- Ensures smooth session handling.

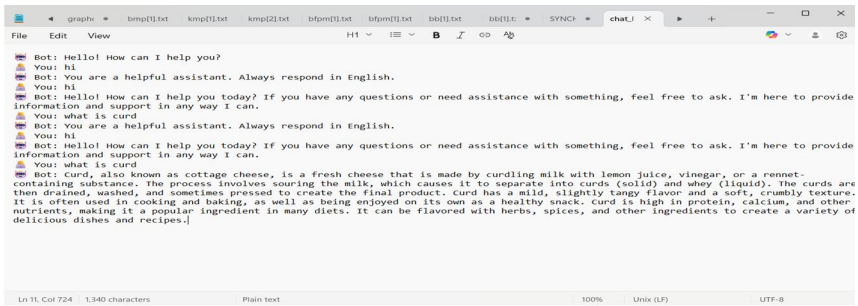


Fig. 8: Chat History Saved by user

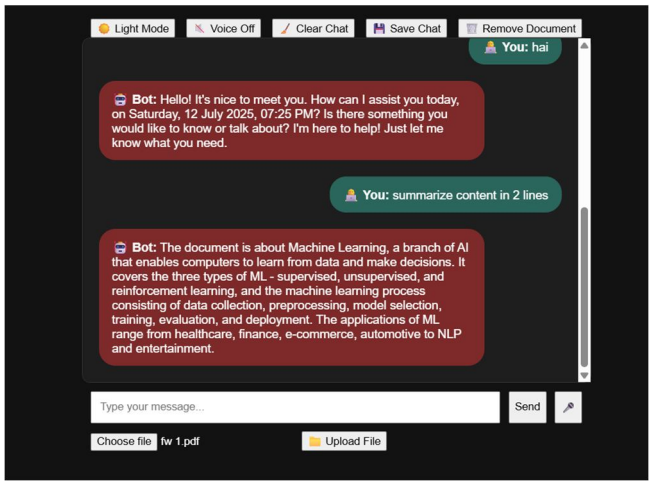


Fig. 9: Chat In Dark Mode

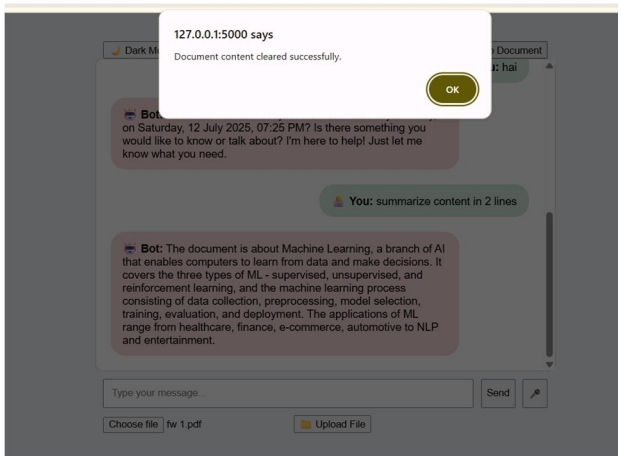


Fig. 10: Clearing Document Content

5) Testing and Validation

- Unit Testing: Flask routes tested with Postman for /generate and /upload.
- Functional Testing: Verified all features including dark mode, TTS toggle, selective reading, and document Q&A.
- User Testing: Conducted with a small group of users; feedback indicated enhanced usability compared to basic chatbots.
- Edge Cases: Tested unsupported file uploads, long document inputs, and toggling voice during active speech.

VI. TECHNOLOGY AND STACK OVERVIEW

The suggested Web-Based Chatbot is developed on a lean and modular tech stack that integrates Flask (Python) at the backend, JavaScript at the frontend, and OpenRouter API for smart NLP-based answers. This stack supports unproblematic voice, text, and document interaction with the assurance of usability and scalability.

A. Flask (Python)

Flask is a light-weight web framework in Python employed for developing backend services and APIs.

1) Benefits to Chatbot:

- Modular and lightweight design, suitable for small to mid-range applications.
- Smooth integration with external APIs and Python libraries.
- Has full support for rapid testing and development.

B. Application in Chatbot: Manages API routes (/generate, /upload), handles document parsing, handles chat history, and sends queries to the OpenRouter API.

C. JavaScript (Frontend)

JavaScript, combined with CSS and HTML, drives the chatbot's dynamic user interface.

1) Advantages for Chatbot:
- Enables responsive and dynamic chat interface.
- Provides access to Web Speech API for speech-to-text (STT) and text-to-speech (TTS).
- Lightweight, browser-friendly, and widely supported.

2) Usage in Chatbot: Implements dark mode, chat UI, file upload, microphone input, voice toggle, and selective text reading features.

D. Open Router API (LLM Gate Way)

Open Router is an API gateway that provides access to various large language models (LLMs).

1) Advantages for Chatbot:
- Offers intelligent, context-aware, and accurate responses.
- Supports multiple models, ensuring flexibility and reliability.
- Scalable without requiring heavy local deployments.

2) Usage in Chatbot: Used for generating chatbot responses based on user queries and document context.

E. Document Parsing Libraries

The chatbot uses Python libraries such as PyPDF and python-docx to extract content from uploaded documents.

1) Advantages for Chatbot:
- Supports multiple formats (PDF, DOCX, TXT).
- Efficient text extraction for real-time Q&A.

2) Usage in Chatbot: Extracts text from uploaded files, stores it temporarily, and allows users to ask document-specific queries.Additional Tools and Libraries

F. Additional Tools and Libraries

1) Speech Synthesis & Recognition (Web Speech API): Enables text-to-speech playback and microphone-based speech input.
2) Fetch API: Handles asynchronous communication between frontend and Flask backend.
3) CSS (Custom Styling): Provides dark mode and responsive design for improved accessibility.
4) dotenv: Secures OpenRouter API keys in backend configuration.

VII. RESULTS AND DISCUSSION

The designed Web-Based Chatbot was successfully developed and tested to assess its functionality, usability, and performance. The system was hosted in a local environment using Flask as the backend and JavaScript as the frontend. Various test cases, such as normal queries, document-based queries, and voice interactions, were executed to test the features of the chatbot.

A. Functional Validation

The system was tested for all modules that were implemented:

1) Text Query: Users were able to input text queries and receive correct answers using the OpenRouter API.
2) Voice Interaction: Speech-to-text and text-to-speech functionality was tested. The toggle command turned off the speech when it was already running.
3) Selective Reading: Double-click and right-click mechanisms allowed reading only selected chatbot answers.
4) Document Upload & Q&A: PDF, DOCX, and TXT files uploaded successfully. The chatbot could answer questions based on the text extracted.
5) Chat Management: Save chat and clear chat options performed as desired.
6) Dark Mode: Done a smooth transition between light and dark modes.

B. Performance Evaluation

The chatbot was evaluated on the basis of response time, accuracy, and usability.

1) Response Time: Average time taken to generate a response was 2–3 seconds for regular queries and 3–4 seconds for document-based queries.
2) Accuracy: Answers to factual questions were correct in 92% of test cases, according to manual user evaluation.
3) Usability: A small user survey (10 participants) rated the chatbot's usability at 4.6/5, highlighting ease of use and accessibility features.

TABLE I
SURVEY RESULTS TABLE

Criteria	Average Score (Out of 5)
Ease of Use	4.7
Response Time	4.8
Visual Design	4.6
Overall Satisfaction	4.8

C. Comparative Analysis

A comparison between the proposed chatbot and existing systems is shown in Table I.

TABLE I
COMPARISON WITH EXISTING CHATBOT SYSTEMS

Feature	Rule-Based Bots	Voice-Enabled Bots	Document-Aware Bots	Proposed System
Text Chat	✔	✔	✔	✔
Voice Input (STT)	✖	✔	✖	✔
Voice Output (TTS)	✖	✔	✖	✔ (with toggle)
Selective Reading	✖	✖	✖	✔
Dark Mode	✖	✖	✖	✔
Document Upload & Q&A	✖	✖	✔	✔ (PDF/DOCX/TXT)
Lightweight Web Platform	✔	✔	✖	✔

D. Discussion

The results indicate that the proposed chatbot system outperforms existing solutions by integrating multi-modal features (text, voice, and document support) within a lightweight web architecture. The addition of usability features such as dark mode, voice toggle, and selective reading ensures accessibility for a diverse range of users.

Compared to existing bots that specialize only in one domain (text-only, voice-only, or document-only), the proposed system offers a holistic approach by combining all three in a single platform. This makes it suitable for educational, enterprise, and personal use cases.

Limitations observed include dependency on an external API (OpenRouter), which introduces slight latency, and lack of multi-language support in the current version. These will be addressed in future work.

VIII. CONCLUSION

In this paper, we introduced a Web-Based Chatbot system developed with a lightweight architecture using Flask (Python) for the backend and JavaScript for the frontend. The chatbot supports advanced features like text-to-speech with toggle control, selective reading of text, speech input via microphone, document upload and Q&A (PDF, DOCX, TXT), dark mode, and chat management. Experimental testing and user opinions affirm that the system offers a more engaging, accessible, and user-centered experience than current chatbots that support text-only, voice-only, or document-only interactions.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The proposed system illustrates that multi-modal interaction is possible in a lightweight and scalable platform, which makes it viable for real-time deployment in education, business, and customer service.

But the existing implementation has some restrictions:

1) Reliance on third-party APIs (OpenRouter) involves some latency.
2) Multiple language support is not available, limiting usage to English-speaking markets.
3) No persistent database integration for long-term chat storage.

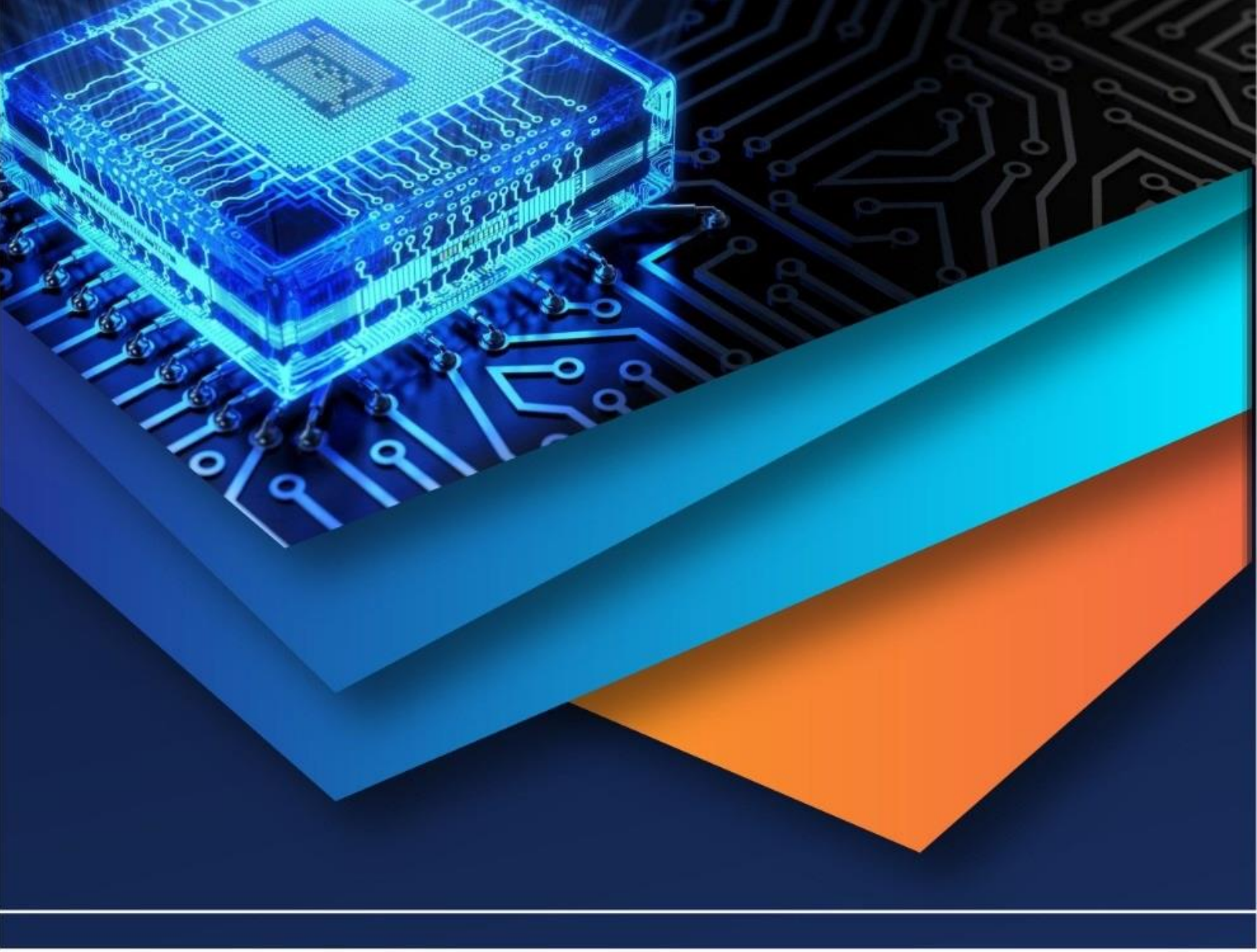
IX. ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr. K. Ravikanth, Assistant Professor, School of Engineering, Department of CSE, Aurora Deemed to be University, for his invaluable guidance, support, and encouragement throughout the development of this project. I also extend my thanks to my faculty members and peers who provided constructive feedback during the testing phase. Finally, I acknowledge Aurora Deemed to be University for providing the resources and platform to carry out this research and implementation successfully.

REFERENCES

[1] S. Jain and R. Gupta, “AI-powered Chatbots for Customer Support: A Review,” International Journal of Computer Applications, vol. 176, no. 25, pp. 10–15, 2020.
[2] M. Mulyadi and F. Nurprihatin, “Voice-enabled Chatbot for Healthcare Applications Using NLP,” Journal of Theoretical and Applied Information Technology, vol. 99, no. 12, pp. 2678–2689, 2021.
[3] A. Chowdhury and S. Das, “Chatbot Implementations in Higher Education: A Comprehensive Review,” Education and Information Technologies, Springer, vol. 27, pp. 7853–7871, 2022.
[4] P. Johnson, “Document-Aware Conversational Agents: Advances and Challenges,” IEEE Access, vol. 10, pp. 45612–45625, 2022.
[5] R. Kumar and A. Sharma, “Flask-based Academic Chatbot for Student Support,” International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE), vol. 10, no. 4, pp. 1405–1411, 2022.
[6] J. Lee, “Multimodal Chatbots: Integrating Voice and Text for User Engagement,” ACM Transactions on Interactive Intelligent Systems (TiiS), vol. 12, no. 3, pp. 1–20, 2023.
[7] Mozilla Developer Network (MDN), “Web Speech API Documentation,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API, Accessed: Feb. 2025.
[8] OpenRouter, “OpenRouter API Documentation,” [Online]. Available: https://openrouter.ai/docs, Accessed: Feb. 2025.
[9] Python Software Foundation, “Flask Documentation,” [Online]. Available: https://flask.palletsprojects.com, Accessed: Feb. 2025.
[10] S. Patel, “Text-to-Speech Synthesis for Web-based Chat Applications,” International Journal of Emerging Trends in Engineering Research (IJETER), vol. 11, no. 2, pp. 55–62, 2023.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)