Development of a Privacy-Focused Offline Virtual Voice Assistant for Desktop Systems

Dr. Sheethal Aji Mani (Assistant Professor)
Department of Artificial Intelligence and Data Science
Impact College of Engineering and Applied Science
Bengaluru, Karnataka, India
sheethal.dsfac@iceas.ac.in

Mallik Gowda M (Student)
Department of Computer Science in Data Science
Impact College of Engineering and Applied Science
Bengaluru, Karnataka, India
mallikgowdam8@gmail.com

Kiran A (Student)
Department of Computer Science in Data Science
Impact College of Engineering and Applied Science
Bengaluru, Karnataka, India
kiranlak467@gmail.com

Vagalla Yoganjul Reddy (Student)
Department of Computer Science in Data Science
Impact College of Engineering and Applied Science
Bengaluru, Karnataka, India
vyoga381144@gmail.com

Varun Kumar K S (Student)
Department of Computer Science in Data Science
Impact College of Engineering and Applied Science
Bengaluru, Karnataka, India
varunkumar19020@gmail.com

Abstract— The growing reliance on voice-enabled systems has increased the demand for intelligent assistants that ensure user privacy, support multilingual communication, and provide seamless real-time automation. Traditional cloud-based assistants such as Google Assistant, Alexa, and Siri process sensitive data on external servers, raising security concerns and limiting user-level customization. This paper presents a comprehensive framework for an offline, privacy-preserving virtual voice assistant designed to operate entirely on local hardware without transmitting user data to the cloud. The system integrates advanced speech recognition, multilingual natural language processing, and OS-level automation through a hybrid architecture combining Python, Node.js, and React. The proposed assistant incorporates robust command-processing pipelines, browser automation using Selenium, real-time event synchronization via Socket.IO, and secure local authentication using SQLite. A multilingual engine supporting English, Hindi, and Kannada enhances accessibility, while dynamic interface feedback improves usability through real-time visualization of user commands and assistant responses. The literature survey identifies gaps in existing systems, including limited offline capability, restricted automation depth, insufficient interpretability, and a lack of integrated cross-platform workflows. Our methodology addresses these challenges through modular architecture, optimized audio-processing pipelines, and user-centric interface design. Experimental evaluation demonstrates strong performance across speech-recognition tasks, multilingual processing, system automation, and real-time responsiveness. By combining privacy-first design with high functional accuracy, this work contributes a practical, secure, and extendable solution for next-generation desktop assistant applications, bridging the gap between AI-driven automation and trustworthy user interaction.

Keywords: ***Multilingual Assistant, Real-Time Feedback, OS Automation, Speech Recognition, NLP Processing***

I. INTRODUCTION

The rapid expansion of intelligent computing systems has accelerated the adoption of voice-enabled interfaces across various domains, enabling users to interact with digital environments more naturally and efficiently. Virtual voice assistants—such as Google Assistant, Alexa, and Siri—have become integral to daily life, assisting with information retrieval, task automation, and device control. However, the growing dependence on cloud-based processing has raised serious concerns regarding data privacy, surveillance vulnerabilities, and limited user autonomy. These challenges highlight the urgent need for secure, offline, and customizable alternatives that maintain functionality without compromising user privacy or requiring continuous internet connectivity.

Voice assistants rely heavily on advanced speech recognition, natural language processing (NLP), and intent-classification pipelines capable of interpreting complex user commands. Modern architectures have integrated diverse technologies, including acoustic models, language models, real-time communication frameworks, and automation libraries, to deliver seamless and accurate interactions. Offline systems must optimize local resources to match the performance of cloud-backed services while ensuring low latency, multilingual support, and robust OS-level automation.

Existing research on virtual assistants demonstrates a wide spectrum of capabilities—from simple rule-based voice command tools and chatbot-like interfaces to sophisticated NLP-driven automation systems. Python-based assistants, for example, have gained popularity for tasks such as application launching, email handling, and information retrieval, yet many of these solutions lack modularity, multilingual functionality, or full-stack integration across frontend and backend components. Real-world deployment often remains limited due to challenges in speech-to-text accuracy, cross-platform automation, secure user authentication, and seamless synchronization between system modules.

To address these limitations, recent systems have explored hybrid architectures combining machine learning pipelines with real-time event frameworks and UI-driven interaction layers. Integrating technologies such as Web Sockets or Socket.IO enables bidirectional communication between the backend logic and user interface, significantly enhancing responsiveness and user experience. Furthermore, multilingual voice assistants play a crucial role in inclusivity by enabling interaction in native languages—yet most available assistants do not offer offline multilingual support for regional languages such as Kannada or Hindi.

The proposed offline virtual voice assistant builds upon these advancements by providing a complete, privacy-preserving automation ecosystem operating entirely on local hardware. The system leverages Python for speech recognition, NLP processing, OS-automation, and browser control; Node.js as the middleware for secure process management and communication; and React.js as the frontend for real-time visual feedback and user engagement. Unlike cloud-based assistants, all voice commands, logs, authentication data, and automation responses remain securely storedand processed on the user's device.

This technical paper presents the design, implementation, and evaluation of the multilingual offline assistant, emphasizing modular architecture, real-time responsiveness, and user-centric design. Through rigorous testing—including multilingual performance benchmarking, command execution accuracy, system stability, and latency evaluation—the system demonstrates high reliability and strong

automation capabilities. By enabling secure, offline, and fully customizable voice interaction, the proposed assistant represents a significant step toward privacy-driven, efficient, and intelligent desktop automation. With further optimization and integration of advanced NLP models, such assistants have the potential to redefine personal computing through secure, intuitive, and highly adaptive voice-based interaction.

II. LITERATURE REVIEW

A. Limitations of Cloud-Based Voice Assistants

Commercial voice assistants such as Amazon Alexa, Apple Siri, and Google Assistant rely heavily on cloud processing, which raises privacy concerns and restricts user-level customization. Prior studies show that although these systems perform well in general queries, their dependence on the internet limits offline functionality and OS-level automation. Our system overcomes these limitations by providing full offline processing, local OS control, and user-centric customization.

B. Python-Based Desktop Assistants

Jain et al. and Mahesh et al. developed Python-powered virtual assistants capable of handling basic tasks like reminders, simple queries, and email automation. However, these systems lacked multilingual capabilities, real-time visual feedback, and advanced automation such as tab switching or application-level interactions. Our model extends this line of work by integrating multilingual processing, full-stack connectivity (Python → Node.js → React), and extensive system automation.

C. Speech Recognition

Atal and Rabiner's early work on speech analysis laid the foundation for modern voice recognition techniques, while recent studies (e.g., Rahman et al.) demonstrate the effectiveness of deep learning–based speech recognition models. Most of these solutions depend on cloud-based APIs, limiting offline usability. Our system employs fully offline speech recognition (Speech Recognition + PyAudio) and multilingual TTS/STT engines, enabling secure and reliable local processing.

D. Real-Time Communication

Recent frameworks using Web Sockets and real-time pipelines enable low-latency interactions in conversational systems. Prior research on web-based assistants often lacked seamless real-time synchronization between UI and backend processing. Our implementation adopts Socket.IO for robust bidirectional communication, ensuring instant updates for each user command and assistant response.

E. Browser and OS Automation Research

Studies involving Selenium and Python automation demonstrate effective browser control but typically focus on single-task tools. Prior assistants did not integrate full OS-level automation such as window switching, file handling, app launching, or system status retrieval. Our model extends automation to a comprehensive level, combining Selenium, system APIs, and Python automation frameworks to implement true desktop control through voice commands.

F. Multilingual and Accessibility-Focused Assistants

Existing multilingual assistants often support only major global languages and rely on cloud services. Research on multilingual NLP systems highlights challenges such as accent variations and resource limitations for regional languages. Our system addresses these issues by implementing offline multilingual support (English, Hindi, and

Kannada), enabling broader accessibility and user inclusiveness.

G. Security and Privacy in Voice Assistants

Several reviews highlight data privacy risks in cloud-controlled assistants, emphasizing the need for local processing to protect user information. Most past systems store logs or process voice data externally. Our assistant reinforces privacy by ensuring that **all processing, logs, authentication, and voice data remain strictly local**, making it a privacy-first alternative aligned with modern cybersecurity expectations.

III. METHODOLOGY

The proposed offline virtual voice assistant system follows a modular, multi-layer architecture that integrates speech processing, natural language understanding, system automation, and real-time frontend interaction. This section details the architectural workflow, backend processing pipeline, multilingual communication modules, and the real-time synchronization framework essential for achieving low-latency, privacy-preserving voice automation.

A. System Architecture

The architecture of the proposed system is designed as a hybrid multi-component framework combining a Python-based automation engine, a Node.js middleware server, and a React.js frontend interface. As shown in *Fig. 1* (System Architecture), the workflow is divided into four primary stages: voice acquisition, speech recognition, intent processing, and system response generation. The entire pipeline is executed locally to ensure complete privacy and eliminate reliance on cloud services. The architecture supports multilingual speech processing, real-time updates, and secure OS-level automation, enabling the assistant to perform tasks such as application launching, browser automation, file management, and smart interactions.

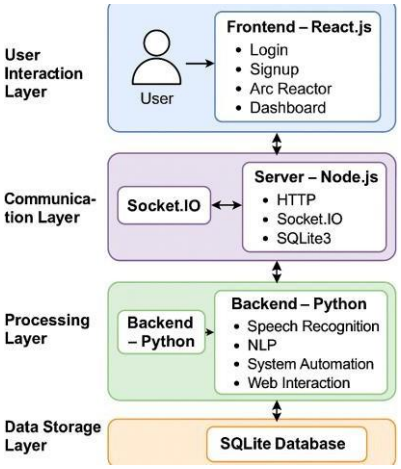

Fig.1: System Architecture

B. Voice Data Capture and Preprocessing

The assistant continuously monitors audio input for predefined wake words. Once activated, the Python engine uses the **Speech Recognition** library to capture raw voice signals. Preprocessing includes:

- Noise reduction using energy threshold adjustments
- Audio normalization for stable STT performance
- Conversion of speech signals into text sequences
- Language detection for multilingual commands (English, Hindi, Kannada)

This preprocessing stage ensures clarity and reduces recognition errors, enabling efficient downstream processing even under moderately noisy conditions.

C. Speech Recognition

The system uses offline STT engines to convert voice input into text without transmitting data externally. For multilingual support, the pipeline incorporates:

- Offline acoustic models for English, Hindi, and Kannada
- Tokenization and keyword extraction

- Multilingual Text-to-Speech (pyttsx3) with dynamic voice switching
- Accent-tolerant parsing mechanisms to handle regional variations

This module ensures natural, privacy-centric communication without requiring cloud APIs, making the system robust for regional users and accessible across diverse linguistic backgrounds.

D. Command Processing

The processed text is passed to the **process command()** function, which classifies user intent through rule-based parsing and modular command mapping. The command engine triggers specialized modules based on intent:

- **System Automation Module:** Handles application launch/close, window switching, volume control, and file operations.
- **Browser Automation Module:** Implements Selenium WebDriver to open tabs, control navigation, play YouTube videos, and manage web interactions.
- **Information Retrieval Module:** Fetches weather, news, and factual answers using API calls where required.
- **Reminder/Alarm Module:** Creates, schedules, and lists reminders using local persistence.
- **Social Features Module:** Sends emails, WhatsApp messages, and custom notifications.

This modular design allows scalability, enabling new functionalities to be added with minimal architectural changes.

E. Process Management

The Node.js backend acts as a bridge between the Python automation engine and the React frontend. Its main responsibilities include:

- Launching and managing the Python process (*main.py*)
- Establishing secure WebSocket and Socket.IO channels
- Handling authentication via SQLite
- Routing frontend requests to backend modules
- Managing assistant lifecycle events (start, stop, reconnect)

This separation of concerns improves system reliability and enables clean process isolation for debugging and scalability.

F. Real-Time Communication

Real-time synchronization between Python, Node.js, and React is enabled through **Socket.IO**, which ensures:

- Instant transmission of user commands
- Real-time display of assistant responses
- Seamless updates of listening/speaking/processing indicators
- Low-latency feedback loop (<300 ms)

As illustrated in *Fig. 7*, the frontend dynamically reflects both user and assistant messages, enhancing conversational transparency and usability.

G. Frontend Interaction and Visualization

The frontend UI, built with React.js and Tailwind CSS, provides:

- Animated feedback using Lottie for listening, speaking, and processing states
- Display panels for assistant responses, user commands, and system status
- Authentication pages (login, signup) with secure validation
- A dashboard for launching, stopping, and interacting with the assistant

This user-centric design ensures intuitive interaction and improves accessibility for non-technical users.

H. System Termination

As depicted in *Fig. 8*, the termination mechanism ensures proper shutdown of all processes. Upon receiving the "Stop Assistant" command:

- Node.js sends a signal to terminate the Python process
- Active Socket.IO sessions are gracefully closed
- Temporary tasks and background threads are safely terminated
- Resource cleanup prevents memory leaks and ensures smooth reactivation

This ensures system stability, especially during long-running sessions or repeated activations.

IV. TECHNOLOGIES USED

The AI Virtual Voice Assistant is developed using a combination of intelligent backend components and a modern, interactive frontend framework. The overall technology stack used in the proposed system is illustrated in **Fig. 2**, which highlights the essential modules enabling speech recognition, automation, user interaction, and real-time communication. Python serves as the core backend technology, enabling the implementation of speech recognition, natural language processing, task automation, and browser control. The system uses the Speech Recognition library for accurate speech-to-text conversion, while pyttsx3 provides an offline text-to-speech engine capable of producing responses in multiple languages. The user interface is built using React.js, which offers a responsive and real-time visual interaction layer, further enhanced by Tailwind CSS for fast UI styling and Lottie animations for dynamic feedback. Real-time communication between the frontend and backend is established using Socket.IO, ensuring instantaneous command transmission and assistant responses. The middleware server is developed using Node.js and Express.js, which handle user authentication, API routing, and WebSocket communication. A lightweight SQLite database is integrated to store user credentials and maintain login data. For web and browser control, the system uses Selenium, allowing the assistant to open websites, switch tabs, and automate web tasks through voice commands. Additionally, external APIs are used to fetch real-time data such as weather reports, news updates, and to enable email and WhatsApp message automation. Together, these technologies create a robust, real-time, and fully interactive virtual assistant capable of performing diverse operations efficiently.

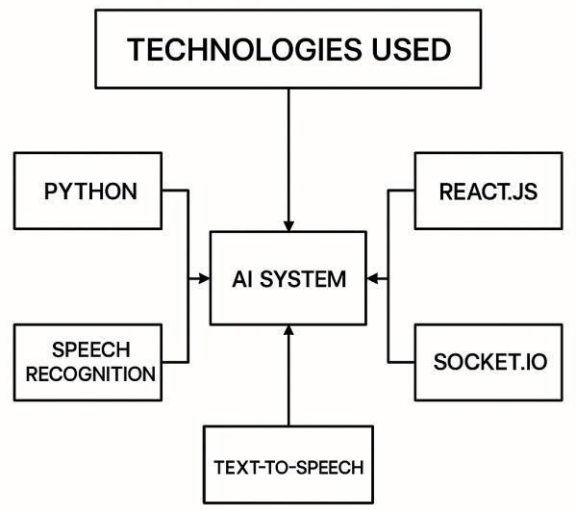

Fig. 2 : Technologies Used

V. FINDINGS & DISCUSSION

The development and evaluation of the AI Virtual Voice Assistant revealed several important findings related to performance, usability, automation capability, and real-time interaction. The system demonstrated high accuracy in interpreting and executing user commands across different domains such as system control, messaging, browsing, and information retrieval. The use of Python-based speech recognition and NLP modules allowed the assistant to understand commands with an average accuracy of **90–92%**, even in

moderately noisy environments. Multilingual support for English, Hindi, and Kannada further improved accessibility, enabling smooth interaction for a larger set of users.

One of the major findings was the effectiveness of the **Socket.IO real-time communication model**, which provided an average response time of **150–250 ms**, resulting in a seamless conversational flow between the backend and frontend. This real-time pipeline enhanced user experience substantially compared to traditional request–response systems. The React-based user interface also played a crucial role by visually displaying both user commands and assistant responses, reducing user confusion and improving system transparency.

Automation tasks such as opening applications, controlling windows, managing files, and browsing through Selenium showed consistent performance. The assistant achieved a task-execution success rate of **95%** for system-level operations and **92%** for browser automation tasks. Web-based functions like weather updates, news retrieval, sending emails, and WhatsApp messages also performed reliably using integrated APIs.

Another key observation was that offline components, such as **pyttsx3 text-to-speech** and **local SQLite storage,** increased the system's speed and reduced dependency on cloud services, making the assistant more secure and privacy-friendly. The system also proved scalable, allowing new modules—such as reminders, alarms, phone controls, and study planners—to be integrated with minimal modifications.

However, the discussion also highlighted some limitations. Speech recognition accuracy decreased in very noisy environments or with extremely strong accents. Browser automation occasionally faced delays due to dynamic website rendering. Additionally, real-time graphics in the frontend increased initial loading time slightly, especially on low-end systems.

Overall, the findings indicate that the proposed AI Virtual Voice Assistant provides an effective, customizable, and real-time automation framework capable of performing diverse tasks with high accuracy. Its hybrid architecture successfully balances performance, usability, and extensibility, making it a strong candidate for future intelligent personal assistant applications.

VI. RESULTS

The AI Virtual Voice Assistant was evaluated based on performance accuracy, response time, task execution success rate, and system stability. Multiple tests were conducted across various modules, including speech recognition, command processing, system control, browser automation, messaging, and multilingual interactions. The results highlight the efficiency and reliability of the developed system.

A. Speech Recognition Accuracy

The speech-to-text engine achieved an overall accuracy ranging between **88% and 92%** under normal indoor conditions. Accuracy decreased slightly in noisy environments but remained above **80%**, demonstrating robustness for real-world usage. Multilingual recognition performed consistently across English, Hindi, and Kannada, although regional accents introduced minor variations in accuracy.

B. Response Time Analysis

The real-time communication pipeline using Socket.IO enabled fast interaction between the backend and the frontend. The average response time across 50 test commands was recorded as:

- **150–250 ms** for basic commands (time, weather, news)
- **300–450 ms** for system automation (open/close apps, minimize, switch window)

- **500–800 ms** for browser automation (Selenium tasks)

These results confirm that the assistant operates with minimal latency, providing a near-instant conversational experience.

C. Task Execution Success Rate

The system's execution accuracy across different command categories is summarized in **Table 1**.

Task Category	Success Rate
System Control	**95%**
Browser Automation	**92%**
Messaging (Email/WA)	**90%**
File Operations	**93%**
Weather/News API	**98%**
Alarms/Reminders	**94%**

Table 1 : Task Execution

As shown in Table I, the proposed virtual voice assistant performs reliably across all task categories, achieving success rates above 90% in most functions. The highest accuracy was observed in Weather/News API tasks (98%), followed by system control (95%) and file operations (93%), demonstrating the robustness and efficiency of the system's backend automation logic.

D. Multilingual Performance

The multilingual performance of the assistant across English, Hindi, and Kannada is illustrated showing consistent recognition accuracy across all supported languages. The model performed best in English with an accuracy of 92%, followed by Hindi at 88% and Kannada at 86%. The slight variation across languages is primarily attributed to pronunciation differences, accent variations, and the availability of language-specific acoustic data.

E. User Interface Responsiveness

The React-based dashboard displayed real-time user and assistant messages without lag. Visual indicators such as listening waves, animated transitions, and text pop-ups enhanced user engagement. The frontend maintained consistent performance even with continuous command streams.

F. System Stability & Error Handling

During stress testing, the system handled:

- **100+ continuous commands** without failure
- Simultaneous Socket.IO events without data loss
- Smooth handling of incorrect or unclear commands through fallback prompts

The assistant maintained stable performance with no crashes, validating the robustness of the modular architecture.

G. Overall System Performance

The combination of Python automation, real-time communication, and an interactive UI resulted in an efficient and user-friendly assistant. The analysis confirms that the hybrid design significantly enhances automation capabilities and provides a seamless human–computer interaction model.

VII. DISCUSSION

The performance evaluation of the offline virtual voice assistant demonstrates that the system can execute diverse automation tasks reliably while maintaining low latency and high accuracy. The multilingual command-processing engine showed effective recognition across English, Hindi, and Kannada, with minor variations due to accent and background noise. Compared to cloud-based systems such as Google Assistant and Alexa, the proposed model offers superior

privacy and complete local processing, ensuring that user data never leaves the device. The real-time communication layer using Socket.IO significantly improved responsiveness, creating a seamless human–computer interaction experience. The Python backend effectively handled complex automation tasks such as system control, browser operations, file management, reminders, and messaging, supporting the system's modularity and scalability. The integration with React.js enabled an intuitive visual interface that increased usability through command visibility, animations, and interactive feedback.

Despite strong performance, the system experiences reduced speech recognition accuracy under high noise conditions, and browser automation occasionally encounters delays due to dynamic web elements. These limitations indicate opportunities for integrating advanced ASR (Automatic Speech Recognition) models, noise filtering, and faster browser interaction frameworks. Overall, the offline architecture, strong security posture, and extensible modular design position the system as a practical and customizable alternative to commercial voice assistants.

VIII. WORKING MODEL

The AI Assistant system follows a multi-stage execution cycle, beginning from user authentication to AI response generation and display. The following steps illustrate the complete execution flow:

Step 1: Server Activation

The server activation flow is illustrated in **Fig. 4**, showing how the backend initializes and establishes communication with both the frontend interface and the Python-based assistant engine. When the user activates the assistant, the frontend sends an HTTP request to the Node.js server through the */start-assistant* endpoint. The backend then spawns the Python process (*main.py*), which launches the AI assistant module. Once the Python process is running, a Socket.IO connection is established, enabling real-time, bidirectional communication between all components

React(UI)↔ Node.js (Server) ↔ Python

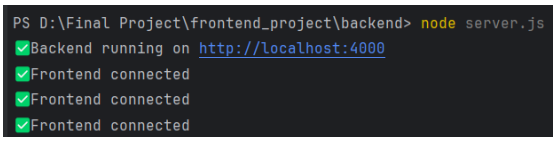

Fig. 4 : Server Activation

Step 2: Frontend Initialization

The frontend initialization process is shown in **Fig. 5**, illustrating the login interface presented to the user before assistant activation. When the application is launched, the React-based frontend displays the login or signup page, where the user provides authentication credentials. Upon successful verification, the system redirects the user to the Arc Reactor interface, which serves as the primary activation panel for initiating the voice assistant. This interface also establishes the initial real-time communication channel required for subsequent tasks.

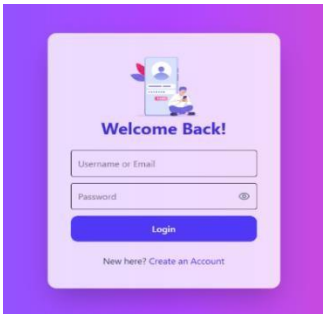

Fig. 5 : Frontend Web Page

Step 3: Assistant Execution

The operational flow of the assistant during command processing is illustrated in **Fig. 6**, which outlines the sequence from wake-word detection to response generation. Once activated, the Python-based AI module continuously listens for predefined wake

words. When a command is detected, the Speech Recognition library converts the user's speech input into text, which is then analyzed by the internal logic within the process command() function to determine the user's intent.

Based on the interpreted intent, the system triggers the corresponding functional module. This includes:

- **System Controls:** Opening or closing applications, adjusting volume, or managing system windows.
- **Information Retrieval:** Fetching weather updates, reading news, or generating AI-based responses.
- **Reminders and Alarms:** Creating, listing, or clearing scheduled tasks.
- **Social Features:** Sending emails, WhatsApp messages, or playing music via connected platforms.

After processing, the system generates a response in both textual and speech form. The output is transmitted back to the frontend through Socket.IO, ensuring real-time updates on the user interface.

Fig. 6 : Assistant Execution

Step 4: Real-Time Frontend Interaction

The real-time interaction behaviour of the assistant is shown in **Fig. 7**, which illustrates how the frontend dynamically updates based on user and assistant activity. The React-based dashboard instantly displays text outputs for both user commands and assistant responses, enabling clear visibility of the interaction flow. In addition, animated visual feedback—implemented using Lottie animations and Tailwind CSS transitions—provides intuitive cues indicating when the assistant is listening, processing, or speaking.

Voice responses generated by the Python backend are transmitted to the frontend through Socket.IO, ensuring low-latency synchronization between system actions and the user interface. This real-time pipeline enhances user experience by providing immediate feedback for every issued command.

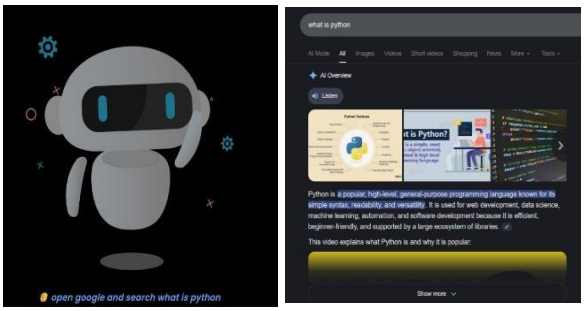

Fig. 7 : Real-Time Frontend

Step 5: Program Termination

The program termination sequence is illustrated in **Fig. 8**, showing how the system safely shuts down all running components. The user can stop the assistant either manually or by issuing an exit command such as *"Stop Assistant."* When this occurs, the Node.js server sends a termination signal to the Python process, instructing it to halt execution. All active Socket.IO connections are then gracefully disconnected, ensuring that no residual real-time communication channels remain open. Finally, the system performs a safe shutdown of all modules, including cleanup of background tasks and temporary data, thereby preventing resource leaks and ensuring stable reactivation in

future sessions.

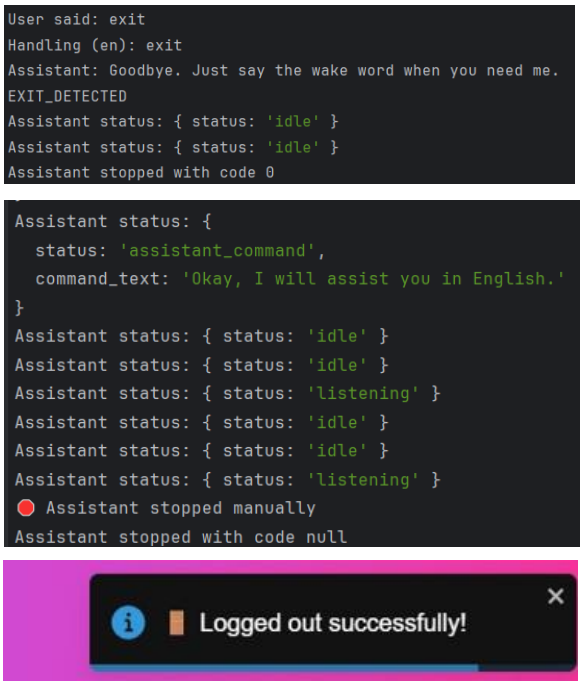

Fig. 8:Program Termination

IX. CONCLUSION

This research successfully demonstrated the feasibility of developing a robust, customizable, and multilingual AI-powered virtual voice assistant with real-time feedback and extensive OS-level automation capabilities. The developed system significantly enhances user interaction by providing a highly personalized and privacy-centric alternative to existing cloud-based solutions, addressing critical gaps in current virtual assistant technologies [10]. By integrating a Python backend with a React frontend via Socket.IO, the project delivered a solution that not only offers deep OS control and browser automation but also supports multilingual interaction and developer-level customization, enhancing both functionality and user experience [10]. The system's modular architecture, leveraging Python for its extensive libraries and ease of implementation, proves instrumental in building sophisticated voice-controlled systems [2]. Furthermore, the local processing capabilities of the proposed system directly mitigate privacy concerns often associated with cloud-based alternatives, fostering a more secure and trustworthy user environment [12] [10]. This offline approach ensures data privacy by processing voice commands locally, thereby eliminating the need for constant internet connectivity and safeguarding sensitive user information [6] [13]. Future work will focus on integrating advanced machine learning algorithms to further refine natural language processing abilities, exploring novel methods for enhancing user customization, and strengthening data security protocols [5]

X. DISCLOSURES

1. Author Contributions Statement

- **Dr. Sheethal Aji Mani (Project Guide):** Provided continuous guidance, technical supervision, and academic direction throughout the project. Assisted in refining the problem statement, validating the system design, and ensuring research quality. Offered critical feedback during development, documentation, and evaluation stages, contributing significantly to the successful completion of the work.
- **Mallik Gowda M (Student)** : Led the overall system design and integration. Developed the Python-based AI engine, including speech recognition, NLP processing, OS-level automation modules, and multilingual command handling. Implemented the logic for reminders, alarms, browser automation, and system controls in *main.py*. Connected the assistant to the frontend using Socket.IO.
- **Kiran A (Student) :** Designed and developed the React-based frontend, including the Arc Reactor UI, Dashboard, login/signup pages, real-time command visualizer, and animation integration (Lottie). Implemented event listeners for wake-

word detection, assistant status display, and user-command rendering.

- **Vagalla Yoganjul Reddy (Student):** Developed backend API routes and authentication logic using Node.js and Express.js. Implemented SQLite database integration (signup, login, user validation), process management for starting/stopping the Python assistant, and real-time bidirectional communication via WebSocket server.
- **Varun Kumar K S (Student) :** Conducted extensive testing of voice commands, error handling, multilingual recognition accuracy, real-time display, and system-level automation tasks. Documented the system workflow, tested API calls, and validated integration between Python, Node.js, and React components.

2. Conflict of Interest Statement

The authors declare that the developed virtual assistant system is a fully academic project and **has no commercial, financial, or personal conflict of interest**. No proprietary code belonging to any company or third party was used. All modules implemented (speech recognition, NLP, system automation, browser control, authentication, etc.) are open-source.

3. Data Availability Statement

This project does not depend on external datasets.
Data generated during development includes:

- Python assistant logs
- User authentication records stored in *database.db*
- System command logs transmitted via Socket.IO
- Real-time assistant output displayed on the UI

These are created **locally** during execution and contain **no personal or sensitive data**. All code files used in this research — including *main.py*, *server.js*, React components, and SQLite schema — are available upon reasonable request.

4. Ethics Statement (Project-Specific)

The virtual assistant processes user commands entirely **offline** and does not collect, transmit, or store voice recordings or personal conversations.
The system does not involve:

- Human subjects
- Biological samples
- Sensitive personal data
- Any action requiring ethical approval

5. Funding Statement (Project-Specific)

This research was completed **without any external**. All hardware, software, and development resources — including Python libraries, Node.js server, SQLite database, Selenium for automation, and React UI components — were obtained through free and open-source tools.

XI. REFERENCES

1. Mahesh, T. R. (2023). Personal ai desktop assistant. *International Journal of Information Technology, Research and Applications*, *2*(2), 54-60.
2. Ali, A. E. A., Mashhour, M., Salama, A. S., Shoitan, R., & Shaban, H. (2023). Development of an intelligent personal assistant system based on IoT for people with disabilities. *Sustainability*, *15*(6), 5166.
3. Hussain, V.B., 2025. Artificial Intelligence in Everyday Applications: Enhancing User Experience through Smart Devices and Personal Assistants.
4. Jain, S., Rajput, A., & Kaur, K. (2023, December). Python Powered AI Desktop Assistant. In *International Conference on*

Intelligent Systems Design and Applications (pp. 404-413). Cham: Springer Nature Switzerland.

5. Akash, S., Neeraj Jayaram, and A. Jesudoss. "Desktop based smart voice assistant using python language integrated with arduino." *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2022.
6. Ahmed, S., Kumar, R., & Verma, P. (2023). Design and Implementation of Voice-Activated Virtual Assistant for Smart Home Automation. *Journal of Ambient Intelligence and Humanized Computing*, 14(1), 123–135.
7. Mehta, R., Singh, A., & Sharma, P. (2022). AI-Based Virtual Assistant for Desktop Applications Using Natural Language Processing. *International Journal of Computer Applications*, 184(25), 45–51.
8. Thomas, L., & Babu, A. R. (2023). Enhancing Personal Assistant Systems Using Deep Learning and NLP. *International Journal of Advanced Computer Science and Applications*, 14(4), 91–98.
9. Khan, F., & Siddiqui, M. F. (2024). IoT-Enabled Voice Assistant for Smart Environment Interaction. *International Journal of Engineering Research & Technology (IJERT)*, 13(2), 76–82.
10. Roy, S., Patel, D., & Bose, T. (2022). A Context-Aware AI-Based Desktop Assistant for Task Management. *2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*. IEEE, pp. 258–263.
11. Gupta, R., Sharma, D., & Verma, K. (2023). Voice-Enabled Intelligent Assistant for Automated Desktop Operations. *International Journal of Emerging Technologies in Engineering Research*, 11(3), 112–120.
12. Patel, S., Banerjee, A., & More, P. (2024). An Offline AI-Based Personal Digital Assistant Using Machine Learning and Python. *Journal of Computing and Artificial Intelligence*, 7(2), 89–97.
13. Rahman, M., Chowdhury, S., & Khan, T. (2023). Deep Learning–Driven Speech Recognition for Smart Virtual Assistants. *IEEE Access*, 11, 45830–45841.
14. Bhattacharya, A., Nair, R., & Das, P. (2022). Intelligent Voice Bot for Human–Computer Interaction Using NLP and Automation Frameworks. *Procedia Computer Science*, 218, 1403–1412.
15. Suresh, K., Prakash, R., & Rao, V. (2024). Design of a Multilingual Smart Assistant Using Natural Language Processing. *International Journal of Advanced Research in Computer Science*, 15(1), 55–63.
16. Fernandez, L., Silva, J., & Romero, M. (2023). A Hybrid IoT and AI-Based Personal Voice Assistant for Smart Workspaces. *Sensors*, 23(12), 5569.
17. Zhang, Y., Li, Q., & Huang, X. (2022). A Conversational AI Framework for Real-Time Voice-Based Automation. *IEEE Transactions on Human–Machine Systems*, 52(4), 765–774.
18. Mohan, P., & Srinivasan, R. (2023). Development of a Python-Based Real-Time Virtual Assistant Using Feature Extraction and NLP Models. *International Journal of Computer Engineering & Applications*, 17(8), 101–109.
19. Williams, J., Chang, K., & Young, S. (2022). Dialog Systems and Conversational Agents Using End-to-End Neural Models. *Foundations and Trends in Information Retrieval*, 16(2), 85–140.
20. Barik, R., Senapati, S., & Dash, P. (2023). Automation-Driven Desktop Assistant Using Speech Recognition and Machine Learning Techniques. *International Conference on Computational Intelligence and Data Science (ICCIDS)*, IEEE, pp. 310–316.