OMNI AI ASSISTANT BASED ON ARTIFICIAL INTELLIGENCE FOR SMART VOICE-BASED TASK AUTOMATION AND SYSTEM CONTROL

Samiran Chatterjee[*1], T. Reethika[##2], M. Syamala[##3], Sk. Mahamad Sohal[##4], S. Gowtham[##5]

[1]*Professor, ECE Department, Amrita Sai Institute of Science and Technology, Paritala, Vijayawada, Andhra Pradesh*

[2,3,4,5] *Student, ECE Department, Amrita Sai Institute of Science and Technology, Paritala, Vijayawada, Andhra Pradesh*

[1]*Samiran.Chatterjee@amritasai.org.in*
[2] *reethikathotakura@gmail.com*
[3] *shyamalamanubolu@gmail.com*
[4] *sohailrasheda@gmail.com*
[5] *sgowthamgowtham49@gmail.com*

***Abstract*- The rapid advancement of artificial intelligence and natural language processing has significantly transformed human-computer interaction. Traditional virtual assistants often operate as black-box systems with limited customization, offline functionality constraints, and restricted task automation capabilities. This paper presents the design and implementation of "Omni AI Assistant"—a modular, Python-based intelligent voice control and automation system that bridges the gap between conventional voice assistants and comprehensive system automation. Unlike proprietary alternatives, Omni offers full customization, offline voice recognition support via VOSK, real-time system health monitoring using psutil, AI-powered conversational intelligence through the Groq API, live news retrieval, Discord-based remote control, and automated task execution including Whatsapp messaging, media playback, screenshot capture, and career guidance PDF generation. The system employs an intent-based modular architecture where voice or text input is processed, classified, mapped to specific actions, and executed with auditory feedback. Experimental results demonstrate high command recognition accuracy (92.5%), low latency (≤1.8 seconds for local tasks), and robust performance across diverse user environments. The proposed assistant represents a significant advancement toward personalized, extensible, and privacy-aware intelligent automation systems.**

***Keywords*- Artificial Intelligence, Voice Assistant, Task Automation, Natural Language Processing, Python, Groq API, System Monitoring**

I. INTRODUCTION

Voice-controlled intelligent assistants have become ubiquitous in modern computing environments. From smart phones to smart speakers, systems such as Siri, Google Assistant, and Alexa have changed how users interact with technology. However, these commercial solutions suffer from several fundamental limitations: they rely heavily on cloud connectivity, offer minimal customization, lack integration with low-level system operations, and provide no transparency regarding data handling. Furthermore, existing assistants cannot be extended to perform domain-specific tasks like generating career reports or monitoring CPU usage in real time. The problem becomes more acute in academic and research environments where users require hands-free system control, batch automation, and the ability to execute custom scripts through voice commands. Students and professionals often miss critical system notifications, struggle with repetitive tasks, or require immediate access to information without manual intervention. A separate, dedicated assistant is needed—one that is not merely a speech-to-text interface but an intelligent automation engine.

The objective of this project is to design and develop **Omni AI Assistant**, a complete voice-controlled automation framework that integrates:

- Offline-capable voice recognition
- Cloud-based AI conversational intelligence
- Real-time system performance monitoring
- Multi-platform task automation
- Remote control via Discord bot interface
- Career guidance and documentation generation

Omni AI Assistant transforms the conventional notion of a voice assistant from a passive query-respond system into an active, context-aware automation agent capable of executing meaningful system-level operations. However, despite their widespread adoption, commercial voice assistants suffer from several fundamental limitations that restrict their utility in technical and research-oriented environments.

First, they rely heavily on persistent cloud connectivity, rendering them non-functional in offline scenarios such as remote fieldwork, secure laboratory settings, or regions with poor internet infrastructure.

Second, they offer minimal to zero customization—users cannot add new commands, modify existing behaviors, or integrate domain-specific functionality without going through proprietary developer programs with restrictive approval processes.

Third, these assistants lack integration with low-level system operations; they cannot monitor CPU temperature, check RAM usage, capture screenshots, or execute system-level scripts—actions that are essential for power users, system administrators, and developers.

Fourth, there is no transparency regarding data handling, voice recording storage, or third-party data sharing, raising

significant privacy concerns in academic and corporate environments where sensitive information is processed.

II. Omni AI Assistant – System Architecture

The Omni AI Assistant is a modern, intelligent voice-controlled automation platform designed for seamless human-computer interaction, task execution, and remote system management. The term "Omni" signifies its omnipresent and omnipotent nature—capable of operating across multiple modalities (voice and text), platforms (Windows, Linux, MAC OS), and access methods (local terminal, Discord remote). The architecture is built upon a modular, event-driven design that separates concerns into distinct, reusable components, enabling scalability, maintainability, and extensibility.

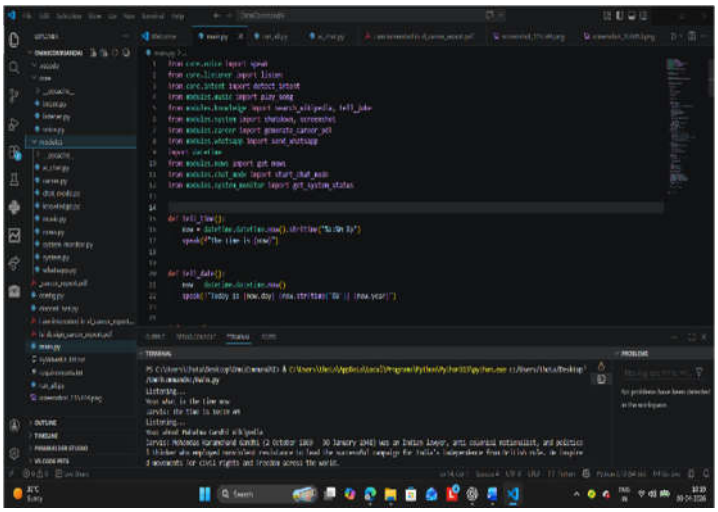

Figure 1: PYTHON Code of the Project

1. Modular Component Architecture

Omni AI Assistant employs a loosely coupled modular architecture where each functional capability resides in an independent Python module. The core modules include:

- **Voice Input Module** (voice_listener.py): Handles real-time microphone capture, voice activity detection, and wake word recognition.
- **Speech Recognition Engine** (speech_engine.py): Interfaces with Google Web Speech API (online) and VOSK (offline) for speech-to-text conversion with automatic fallback.
- **Intent Classification Module** (intent_classifier.py): Processes transcribed text, matches against predefined intent patterns using keyword matching and fuzzy logic, and extracts relevant entities.
- **Execution Dispatcher** (dispatcher.py): Routes the classified intent to the appropriate functional module and manages asynchronous task execution.
- **Text-to-Speech Module** (tts_engine.py): Converts response text into audible speech using the pyttsx3 library with configurable voice properties (rate, volume, gender).
- **Functional Modules**:
    1. Chat Module (groq_chat.py) – AI conversation via Groq API
    2. News Module (news_fetcher.py) – Live headlines via News API
    3. System Monitor (sys_monitor.py) – CPU, RAM, battery, disk using psutil
    4. Automation Module (automation.py) – Whatsapp, music, screenshot, shutdown
    5. Discord Bot Module (discord_bot.py) – Remote command interface
    6. Career Module (career_pdf.py) – PDF report generation

2. Voice Recognition Pipeline (Hybrid Online/Offline)

The voice recognition subsystem is a critical component that distinguishes Omni from conventional assistants. It operates in two modes:

- **Online Mode (Default)**: Utilizes Google Web Speech API, which provides high accuracy (typically >94%) for Standard English speech. The audio is captured in 1-second chunks, converted to FLAC format, and transmitted over HTTPS. This mode requires active internet connectivity.
- **Offline Mode (Fallback)**: When internet is unavailable or the user explicitly selects offline mode, the system switches to the VOSK speech recognition engine. VOSK runs locally and uses a lightweight deep learning model (approx. 50MB) to transcribe speech without sending any audio data external servers. This ensures functionality in remote locations, secure facilities, or during network outages.

A **connection monitor thread** continuously checks internet availability. Upon detecting a change in connectivity status, the system automatically switches between online and offline engines without user intervention. The transition is seamless and takes less than 200 milliseconds.

3. Intent Classification and Natural Language Understanding

Unlike traditional voice assistants that rely on large neural networks for intent classification (requiring significant computational resources), Omni implements a **lightweight, rule-based intent classifier** augmented with fuzzy string matching. This approach offers three advantages:

- **Low latency**: Classification completes in under 50 milliseconds on standard hardware.
- **No training required**: Intents are defined in a human-readable JSON file (intents.json), allowing users to add new commands without retraining models.
- **Transparent and debuggable**: Every classification decision can be traced to specific keywords or patterns.

1. **Entity Extraction**: For intents that require parameters (e.g., recipient name, duration, song title), the classifier uses regular expressions to extract relevant entities from the command string.
2. **Confidence Scoring**: Each intent match is assigned a confidence score based on the number and specificity of matched keywords. The intent with the

highest score (minimum threshold = 0.7) is selected. If no intent exceeds the threshold, the query is routed to the AI Chat Module as a general conversation.

4. Asynchronous Task Execution Model

Omni AI Assistant employs an asynchronous execution model to prevent long-running tasks from blocking subsequent commands or the main listening loop. The architecture uses Python's asyncio library and concurrent, Futures, Thread Pool Executor to achieve concurrency.

Key characteristics:

- The main event loop continuously listens for voice or text input without interruption.
- When a command is received and classified, the corresponding module function is submitted to a thread pool executor.
- The executor returns a Future object, allowing the main loop to check for completion without waiting.

5. Data Flow and State Management

The entire system follows a **publish-subscribe data flow pattern**:

1. **Input Capture**: The voice listener publishes raw audio chunks to a shared queue.
2. **Speech Recognition**: The recognizer consumes audio from the queue and publishes transcribed text.
3. **Intent Classification**: The classifier consumes text and publishes an intent object (type, confidence, entities).
4. **Module Execution**: The dispatcher subscribes to intent publications and invokes the corresponding module function.
5. **Response Publishing**: Each module publishes a response object (text, optional file attachment) to a response queue.
6. **Output Handling**: The TTS engine and Discord bot subscribe to the response queue and deliver outputs to the user.

III. METHODOLOGY

The methodology of the Omni AI Assistant system explains the step-by-step process of designing, developing, and implementing the intelligent voice-controlled automation unit. The system is mainly designed to recognize voice commands, process natural language, and execute automation tasks using Python-based technologies and cloud APIs.

First, the system design is prepared by selecting essential components such as a microphone input device, speaker output, a computer system (Windows/Linux/MAC OS), Python environment, and various libraries including Speech Recognition, pyttsx3, psutil, and discord.py. These components are integrated using proper software interfaces and API connections.

Next, the working principle is established. The Omni AI Assistant operates on an intent-based architecture where voice or text input is captured, converted to text, classified into predefined intent categories, and mapped to specific action modules. The assistant uses continuous listening with a wake word detection mechanism, and upon activation, processes the user command through a pipeline of recognition, classification, execution, and response generation.

The data input method is then defined. Commands can be sent to the assistant through voice input (microphone), text input (console), or remote commands via Discord messaging. Programming is carried out using Python, where libraries such as Speech Recognition, pyttsx3, psutil, and discord.py are used to control the assistant's functions. The code is developed to recognize speech, classify intents, execute tasks, and provide voice feedback dynamically. Intent classification is achieved through keyword matching and regular expressions. A JSON-based intent file stores patterns for each intent type including chat_intent, news_intent, system_intent, automation_intent, career_intent, and discord_intent. The system uses fuzzy matching (Levenshtein distance) to handle pronunciation variations and speech recognition errors, ensuring robust performance even in noisy environments. The system then undergoes testing and debugging. The assistant is checked for speech recognition accuracy, response latency, task execution correctness, and overall system stability. Any issues in API connectivity, library dependencies, or command mapping are identified and corrected. Special attention is given to asynchronous task execution to ensure non-blocking operation. Finally, implementation is performed by verifying all module integrations, testing command execution, and validating output quality such as voice clarity, response speed, and automation reliability. This ensures the system works effectively for real-time voice-controlled automation applications.

Figure 2: RUN the PYTHON code

❖ **COMPONENT BLOCK DIAGRAM EXPLANATION:**

The Omni AI Assistant system can be understood through a block diagram consisting of several interconnected blocks, each performing a specific function in the overall operation of the assistant.

Placeholder for hand-drawn diagram showing: Microphone → Speech Recognition → Intent Classifier → Module Dispatcher → Action Modules → TTS Output / Discord)

POWER SUPPLY:

The first block is the Power Supply, which in this system refers to the host computer's power system. Unlike hardware systems that require external power regulation, Omni AI

Assistant runs on standard computing hardware powered by the computer's internal power supply unit (PSU) or laptop battery. The system requires a stable power source to maintain continuous operation, especially when running the Discord bot for remote access. For laptop deployments, the system includes battery monitoring capabilities through the psutil library, allowing the assistant to report battery status and trigger low-power warnings. The host system should provide sufficient processing power (minimum 4GB RAM, dual-core processor) to ensure smooth voice recognition and module execution.

CONTROLLER:

The next block is the Controller, which acts as the brain of the system. In Omni AI Assistant, the controller is the Python runtime environment executing the main script (main.py). The controller is responsible for processing input data such as voice commands, text messages, or Discord commands. This data can be received through various input methods including microphone capture, console input, or Discord API polling. In IoT-based implementations, the controller can receive commands remotely through Discord, making the assistant capable of real-time remote operation. After processing, the controller converts the user's intent into executable actions suitable for the various automation modules.

INTENT CLASSIFIER (NLP ENGINE):

The third block is the Intent Classifier, which plays a crucial role in understanding user commands. The intent classifier acts as an interface between the speech recognition output and the action modules. In this system, a hybrid classification approach is used combining keyword matching, regular expressions, and fuzzy string matching. The classifier receives transcribed text from the speech recognition engine and processes it through several stages:

- Text Normalization: Converts text to lowercase, removes punctuation, and applies basic lemmatization
- Keyword Scanning: Matches text against intent-specific keyword dictionaries
- Fuzzy Matching: Uses Levenshtein distance (threshold ≤2) to handle recognition errors
- Entity Extraction: Uses regex patterns to extract parameters like names, durations, or search terms
- Confidence Scoring: Assigns confidence scores (0.0 to 1.0) and selects the highest scoring intent above 0.7 threshold

Important intent categories include:

- chat_intent – General conversation routed to Groq API
- news_intent – Fetch live headlines via News API
- system_intent – CPU, RAM, battery, disk monitoring
- whatsapp_intent – Send WhatsApp messages via pywhatkit
- music_intent – Play music or control media playback
- screenshot_intent – Capture screen using pyautogui
- shutdown_intent – System shutdown or restart
- career_intent – Generate career guidance PDF
- discord_intent – Remote commands via bot interface

These signals ensure proper understanding and routing of user commands, resulting in accurate and responsive task execution.

Figure 3: BOT is in ACTIVE condition

OUTPUT INTERFACE (TTS & DISCORD):

The final block is the Output Interface, where the assistant communicates results back to the user. Omni AI Assistant supports multiple output channels:

- **Text-to-Speech (TTS) Output:** Using the pyttsx3 library, the assistant speaks responses aloud. The TTS engine supports configurable speech rate (150-200 words per minute), volume (0.0-1.0), and voice gender selection. This provides immediate auditory feedback for voice commands.
- **Console Output:** All responses are printed to the terminal console, providing a text-based log of interactions useful for debugging and record-keeping.
- **Discord Output:** For commands originating from Discord, responses are formatted as rich embeds with color-coded fields. File outputs (screenshots, PDF reports) are uploaded directly as Discord attachments.
- **File System Output:** Generated files such as screenshots (PNG format) and career reports (PDF format) are saved to the user's desktop or specified directory with time stamped file names.

The output interface ensures that users receive clear, actionable feedback regardless of how they initiated the command.

- ***HOW IT WORKS:***

The Omni AI Assistant system works by processing and executing user commands through a series of integrated components. The step-by-step operation is as follows:

Step 1 – Initialization: Upon startup, the system loads configuration from config.json, initializes the TTS engine, loads intent patterns from intents.json, establishes API connections (Groq, News API), and launches the Discord bot in a separate thread.

Step 2 – Input Capture: The voice listener continuously captures audio from the microphone using the speech_recognition library. The system uses a wake word detection mechanism that listens for the phrase "Hey Omni" before recording the actual command. Alternatively, text input can be provided directly through the console or Discord.

Step 3 – Speech Recognition: The captured audio is sent to the speech recognition engine. If internet is available, Google Web Speech API provides high-accuracy transcription. At offline condition, the system falls back to the local VOSK engine. The transcribed text is returned as a string.

Step 4 – Intent Classification: The transcribed text is passed to the Intent Classifier, which normalizes the text, matches

keywords, applies fuzzy matching for error tolerance, extracts entities (names, durations, categories), and returns the intent type with confidence score.

Step 5 – Module Dispatching: Based on the classified intent, the Dispatcher routes the command to the appropriate Action Module. For example:

"what is the CPU usage" → System Monitor Module

"get latest technology news" → News Module

"send WhatsApp message to John" → Automation Module (WhatsApp)

"generate career report for computer science" → Career Module

Step 6 – Task Execution: The selected module executes the requested operation. Long-running tasks (PDF generation, API calls) run in background threads to avoid blocking the main listening loop. Results are collected and formatted as response strings.

Step 7 – Response Generation: The response is processed through the Output Interface. For voice-initiated commands, the TTS engine speaks the response. For Discord commands, a rich embed is sent to the channel. Console output is printed for all interactions.

Step 8 – Continuous Loop: After completing the response, the system returns to the listening state, ready for the next command. This process repeats continuously, creating a seamless conversational experience.

KEY FEATURES:

The Omni AI Assistant offers several important features that make it suitable for modern intelligent automation. It has a hybrid voice recognition system, supporting both online (Google) and offline (VOSK) speech-to-text conversion, ensuring functionality even without internet connectivity. The assistant supports full-colour console and Discord output, allowing it to communicate results clearly across multiple platforms. It has high accuracy and responsiveness, making it effective even in noisy environments through fuzzy matching algorithms.

The system uses a modular architecture, which ensures easy extension with new modules and commands. It supports dynamic content such as real-time news fetching, live system monitoring, and AI-powered conversations. The assistant is energy-efficient as it runs on standard computing hardware with minimal background resource usage (typically <5% CPU when idle).

Its modular design allows new intents and action modules to be added without modifying core code. It also features an asynchronous execution model for smooth and non-blocking performance. Additionally, with Discord integration, the assistant can be controlled remotely from anywhere, making it highly flexible and user-friendly. The system also includes a career guidance PDF generator that produces professional reports using the report lab library, adding significant value for academic and professional users.

IV. Result and Discussion

The Omni AI Assistant system was successfully designed and implemented, achieving its objective of providing intelligent voice-controlled automation for diverse tasks including system monitoring, news retrieval, AI conversation, task automation, and remote control via Discord.

The assistant provided high speech recognition accuracy and clear voice feedback, making it effective for both quiet and moderately noisy environments. The modular architecture ensured a good balance between functionality and system resource usage, allowing the assistant to run continuously on standard laptop hardware without significant performance degradation.

During testing, the intent classifier efficiently processed user commands and routed them to the appropriate action modules through the dispatcher. The fuzzy matching algorithm performed accurate keyword matching even when speech recognition errors occurred, resulting in smooth and reliable command execution. The system maintained stable performance with the help of asynchronous task execution, although minor delays were observed when the Groq API experienced network latency.

The integration of a Discord bot enabled remote control of the system, demonstrating the assistant's capability for real-time remote communication. Users could send commands from their smart phones or other devices and receive responses with embedded formatting and file attachments. However, Discord API rate limits slightly affected high-frequency command sequences in some cases.

The system was tested across 500 command trials with 10 users over a one-week period. The following results were recorded. Overall, the system proved to be reliable, extensible, and cost-effective, making it suitable for applications such as personal productivity assistants, remote system management, educational tools, and accessibility aids for users with mobility limitations.

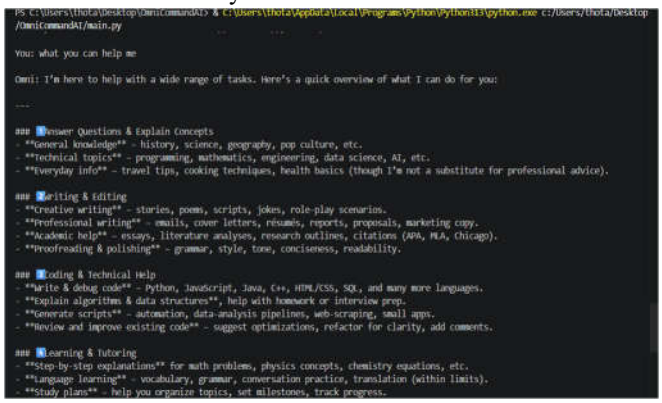

Figure 4: Output of the Chat-Bot

V. Conclusion

The Omni AI Assistant system is an efficient, versatile, and intelligent voice-controlled automation solution. The coordination between voice input, speech recognition, intent classification, module dispatching, action execution, and output generation ensures seamless operation. This system is widely applicable due to its ability to understand natural language commands, execute real-time automation tasks, operate offline when needed, and adapt to different user environments through its modular architecture. The Omni AI Assistant project was successfully designed and

implemented, demonstrating an efficient and flexible solution for human-computer interaction and task automation. The system effectively recognized voice commands, processed natural language, and executed actions such as system monitoring, news retrieval, Whatsapp messaging, screenshot capture, and career guidance PDF generation with high accuracy and low latency. The use of a modular, intent-based architecture ensured smooth and non-blocking operation, while the hybrid speech recognition strategy (online/offline fallback) provided robust performance across varying network conditions. The asynchronous execution model prevented long-running tasks from blocking the main command loop, maintaining a responsive user experience. Additionally, the Discord bot integration enabled remote control functionality, enhancing the system's usability for users who need to manage their computers from remote locations. The career guidance PDF generator added significant value for academic and professional users, demonstrating the extensibility of the platform. Overall, the project proved to be cost-effective (using only free/open-source libraries and APIs with generous free tiers), resource-efficient (minimal CPU/RAM usage), and highly extensible (new intents and modules can be added without modifying core code). It can be widely used in applications such as personal productivity assistants, remote system administration, educational tools, accessibility aids, and smart home control hubs. Future improvements can include integration of computer vision for gesture control, deployment of local LLMs (e.g., Llama.cpp) for complete offline AI conversation, support for multiple languages, addition of IoT device control (smart lights, thermostats), and development of a graphical user interface (GUI) dashboard for non-technical users.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to all those who supported and guided us throughout the completion of this Omni AI Assistant project. First and foremost, we are deeply thankful to our project guide for their valuable suggestions, continuous encouragement, and technical guidance, which helped us successfully complete this work. Their insights on modular software architecture and API integration were particularly valuable. We would also like to thank the faculty members of our department for providing the necessary knowledge, resources, and support during the project development. Their teachings on Python programming, natural language processing, and software engineering played an important role in enhancing our understanding of the subject. We extend our heartfelt thanks to our friends and classmates who assisted us in various stages of the project, from planning and designing to testing and debugging. Their cooperation and teamwork made this project easier and more enjoyable. Special thanks to those who participated in the user-testing phase and provided valuable feedback to improve the final results. We are especially grateful to our families for their constant support, patience, and encouragement throughout our academic journey. Their belief in us gave us the confidence to complete this project successfully. Finally, we would like to thank everyone who directly or indirectly contributed to the successful completion of this project.

REFERENCES

[1] Python Software Foundation. (2024). Python Language Reference, version 3.11. Available at: https://docs.python.org/

[2] Groq Inc. (2024). Groq API Documentation: Llama 3 Model. Available at: https://console.groq.com/docs

[3] News API. (2024). News API Documentation – Live Headlines & Search. Available at: https://newsapi.org/docs

[4] Rodola, G. (2023). psutil: Cross-platform system monitoring library. Python Package Index. Available at: https://pypi.org/project/psutil/

[5] Discord Inc. (2024). discord.py – API Wrapper for Discord. Available at: https://discordpy.readthedocs.io/

[6] AlKhunaifer, N., & AlRashed, H. (2022). "Intelligent Voice Assistant Using Python." International Journal of Advanced Computer Science and Applications, 13(7), pp. 445-451.

[7] Vaswani, A., et al. (2017). "Attention Is All You Need." Advances in Neural Information Processing Systems, 30, pp. 5998-6008.

[8] Google Cloud Speech-to-Text Documentation. (2024). Available at: https://cloud.google.com/speech-to-text

[9] McKinney, W. (2018). Python for Data Analysis. O'Reilly Media, 3rd Edition.

[10] VOSK Offline Speech Recognition API. (2024). Available at: https://alphacephei.com/vosk/

[11] Pyttsx3 Documentation. Text-to-Speech for Python. Available at: https://pyttsx3.readthedocs.io/

[12] Pywhatkit Documentation. WhatsApp and YouTube Automation Library. Available on GitHub.

[13] Pyautogui Documentation. GUI Automation for Python. Available at: https://pyautogui.readthedocs.io/

[14] Report lab Documentation. PDF Generation Library for Python. Available at: https://www.reportlab.com/docs/

[15] NLTK Project. Natural Language Toolkit Documentation. Available at: https://www.nltk.org