iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH
IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 Issue: XII Month of publication: December 2025
DOI: https://doi.org/10.22214/ijraset.2025.76641

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 XII Dec 2025- Available at www.ijraset.com

AI-Based Voice Code Assistant

Mr. Nitin Jain[1], Kushala S[2], Manasvi V Shetty[3], Sahana R[4], Sandhya P[5]
[1]*Associate Professor, Dept. of Information Science and Engineering, Bahubali College of Engineering, Shravanabelagola, Hassan, Karnataka, India*
[2, 3, 4, 5]*UG Student, Dept. of Information Science and Engineering, Bahubali College of Engineering, Shravanabelagola, Hassan, Karnataka, India*

Abstract: Voice-driven human–computer interaction has gained significant attention in recent years due to its potential to improve accessibility and usability. Programming, however, still largely depends on manual typing and complex development environments, which can be challenging for beginners and users with physical limitations. This paper presents an AI-Based Voice Code Assistant, an offline system that enables users to interact with a programming environment using natural language voice commands. The proposed system allows users to load programs, execute code, and obtain explanations through voice or text input within a single integrated platform. It supports multiple programming languages and operates without internet connectivity, ensuring privacy and reliability. Experimental results demonstrate that the system reduces typing effort, improves learning efficiency, and provides an interactive coding experience suitable for educational environments.
Keywords: Voice Programming, Speech Recognition, Offline AI, Programming Education, Accessibility, Code Execution.

I. INTRODUCTION

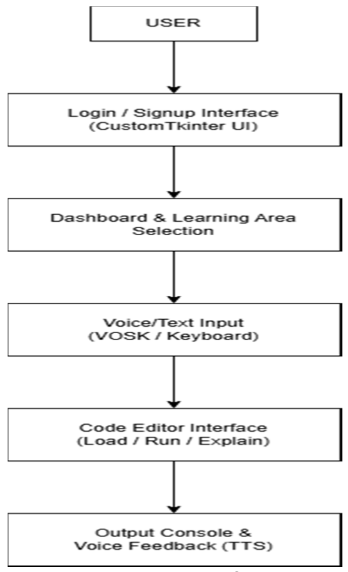
Programming education often requires learners to interact with complex development tools and strict syntax rules, which can slow down the learning process and increase frustration among beginners. Conventional integrated development environments rely on continuous keyboard usage, making programming difficult for users with limited typing skills or physical impairments. With advancements in artificial intelligence and speech recognition, voice-based interaction has emerged as an effective alternative for human–computer interaction. Existing voice-enabled tools, however, are mostly cloud-dependent and lack integrated facilities for code execution and explanation. This paper presents an AI-Based Voice Code Assistant that enables offline voice-driven programming and provides an interactive learning environment for students and developers. The motivation behind this work is to minimize dependency on manual typing and to provide an accessible programming environment that supports voice interaction while maintaining user privacy through offline operation. This work contributes an offline voice-based programming assistant that integrates speech recognition, code execution, and explanation features into a single platform suitable for academic and learning environments.

II. RELATED WORK

Several studies have explored programming-by-voice to improve accessibility and usability. Early systems focused on direct speech-to-text conversion, which resulted in high error rates and increased vocal strain. Later approaches introduced structured voice commands and syntax-directed programming to improve recognition accuracy. Recent research integrates voice interaction into development environments to assist users with disabilities and beginners. Despite these advancements, most existing systems depend on cloud services, provide limited real-time execution support, and lack offline functionality, which motivates the proposed solution.

III. PROPOSED SYSTEM

The proposed AI-Based Voice Code Assistant is a standalone desktop application that supports both voice-based and text- based programming interactions. The system architecture is divided into two major components, namely the client module and the server or core processing module. The client module is responsible for managing all user interactions and presentation-level functionalities. It provides graphical user interfaces for login and signup, dashboard navigation, learning area selection, code editing, and output visualization. The client module captures user input either through voice commands or keyboard input and forwards user requests such as loading programs, executing code, and requesting explanations to the server module for processing. In addition, it provides voice feedback to the user using text-to-speech technology.


USER
Login / Signup Interface (CustomTkinter UI)
Dashboard & Learning Area Selection
Voice/Text Input (VOSK / Keyboard)
Code Editor Interface (Load / Run / Explain)
Output Console & Voice Feedback (TTS)
Fig. 1 Block Diagram of Client Module

During system operation, the user first logs into the application and selects the required learning or programming area. The user then issues commands using either voice input or text input. Based on the issued command, the system processes the request and executes the corresponding operation. The program output and explanations are displayed through the graphical interface, and voice feedback is provided to guide the user throughout the interaction.

The server or core processing module handles all backend operations of the AI-Based Voice Code Assistant. It performs secure user authentication by validating user credentials and manages command processing and program execution. The server module converts voice input into text using speech recognition techniques and extracts relevant command keywords to determine the requested operation. Based on the interpreted command, the module executes the requested program using appropriate compilers or interpreters and maintains execution logs. The processed results, including execution output and explanations, are then sent back to the client module for display and user feedback.

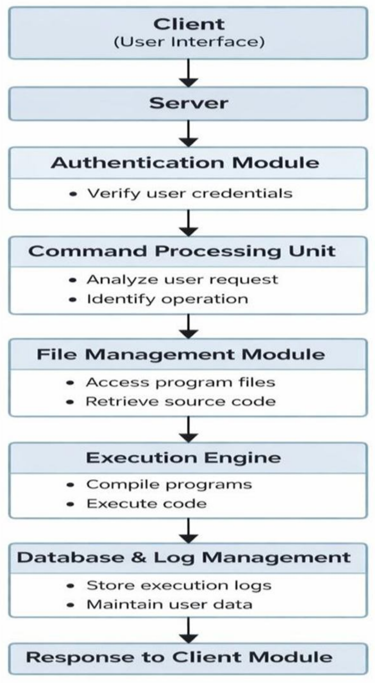


Fig. 2 Block Diagram of Server Module

The below client module functional requirements table describes the features available to the end user of the AI-Based Voice Code Assistant. It explains how users interact with the system through login, learning area selection, and program handling options. The table clearly shows that the client module allows users to load programs, execute code, and request explanations using either voice commands or text input. It also highlights additional support features such as access to learning materials and secure logout, which together ensure a smooth and user-friendly interaction with the system.

TABLE I
CLIENT MODULE FUNCTIONAL REQUIREMENTS

User (student/Developer)	Functionalities
	▪ Requests access to the system by logging in using a Username and Password.
	▪ Creates a new user account with secure password validation.
	▪ Selects learning areas such as ADA, DSA, C Lab, Python Lab, Java Lab, or CPP Lab.
	▪ Loads programming code files by typing or speaking the program name.
	▪ Executes code programs using Run command through GUI or voice input.
	▪ Requests program explanation using Explain command via voice or button click.
	▪ Accesses study materials and reference documents from the dashboard.
	▪ Logs out securely from the system.

The server or core processing module functional requirements table describes the internal operations that enable the AI- Based Voice Code Assistant to function efficiently and reliably. It explains how the backend module performs secure user authentication by validating credentials stored in the database before granting access to the system. The table also outlines the process of interpreting user voice commands by converting speech into text and extracting relevant keywords to determine the requested action. In addition, the table highlights how the system handles program-related operations such as loading source files, compiling code, and executing programs written in multiple programming languages using appropriate compilers and interpreters. Runtime control mechanisms manage program input and output, monitor execution flow, and handle errors during compilation or execution. Furthermore, logging and monitoring functions record execution details, missing program requests, and system errors, which help in tracking system behavior, improving reliability, and supporting future system enhancements.

TABLE II
SERVER MODULE FUNCTIONAL REQUIREMENTS

SL No.	Functionalities
1	▪ The system validates user credentials using a SQLite database with encrypted password storage.
2	▪ The server module processes voice input, extracts keywords, and maps them to available program files.
3	▪ The system compiles and executes programs written in C, C++, Python, and Java using internal compiler and interpreter configurations.
4	▪ The server monitors program execution, handles timeouts, and manages input/output streams.
5	▪ Missing program requests are logged with timestamp, user details, and requested keyword for future enhancement.

IV. SYSTEM ARCHITECTURE AND METHODOLOGY

The proposed AI-Based Voice Code Assistant follows a modular architecture to ensure scalability, maintainability, and ease of future enhancement. By dividing the system into independent modules, each component can be developed, tested, and updated without affecting the overall system. This design approach improves reliability and allows new features, programming languages, or tools to be integrated easily.

A. Architecture Overview

The system architecture consists of a client interface that interacts with a core processing unit. The client interface is responsible for handling user interactions such as login, learning area selection, command input, code editing, and output display. The core processing unit performs backend operations and includes modules for speech recognition, command interpretation, program execution, data storage, and logging. Speech recognition modules convert voice input into text, while execution engines compile and run programs using appropriate compilers or interpreters. Storage modules manage user data, program files, and study materials, and logging mechanisms record execution details and errors. Communication between the client interface and the core processing unit ensures smooth request handling and real-time feedback.

B. Methodology:

The development methodology begins with requirement analysis, where functional and non-functional requirements are identified based on user needs and academic programming environments. This is followed by system design, where architectural diagrams, flow diagrams, and sequence diagrams are used to model the system structure and interaction flow. After design finalization, the system is implemented using Python along with offline AI tools for speech recognition and text-to- speech. Each module is developed independently and then integrated into the complete system. Finally, testing and validation are carried out to verify correct functionality, performance, and reliability. This systematic development process ensures that the AI-Based Voice Code Assistant operates efficiently and meets its intended objectives. The below sequence diagram shows the overall workflow of the AI-Based Voice Code Assistant. The user installs and opens the application, logs in, and accesses the dashboard. The user selects a programming area, opens the editor, loads or modifies code, and runs the program. The system compiles the code, displays the output or errors, accepts input if required, and confirms successful execution.

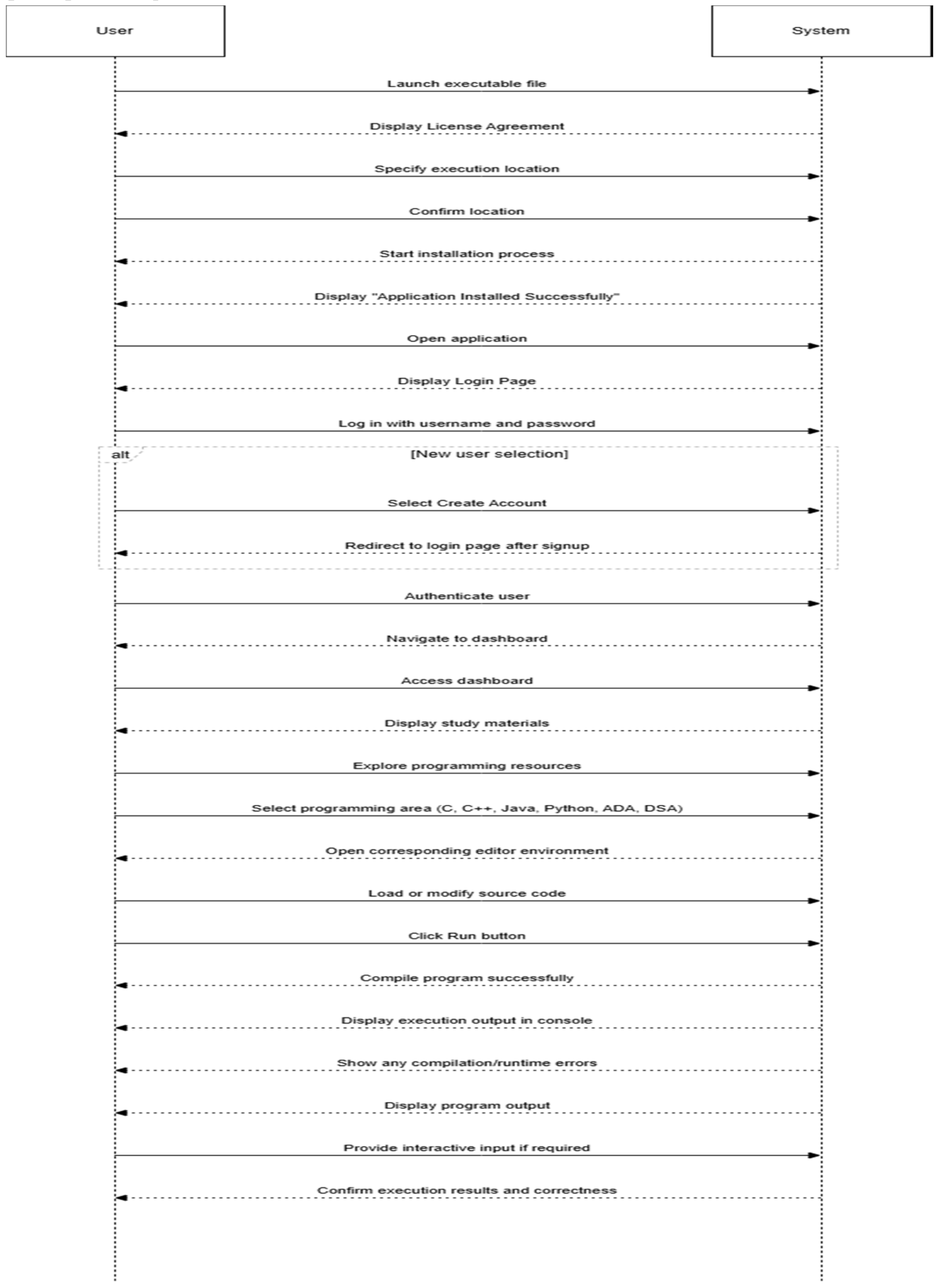

User
System
Launch executable file
Display License Agreement
Specify execution location
Confirm location
Start installation process
Display "Application Installed Successfully"
Open application
Display Login Page
Log in with username and password
alt
[New user selection]
Select Create Account
Redirect to login page after signup
Authenticate user
Navigate to dashboard
Access dashboard
Display study materials
Explore programming resources
Select programming area (C, C++, Java, Python, ADA, DSA)
Open corresponding editor environment
Load or modify source code
Click Run button
Compile program successfully
Display execution output in console
Show any compilation/runtime errors
Display program output
Provide interactive input if required
Confirm execution results and correctness

Fig. 3 Sequence Diagram of Client-System Interaction

V. IMPLEMENTATION DETAILS

The system is implemented using Python as the primary programming language. CustomTkinter is used to develop a modern graphical user interface. Offline speech recognition is implemented using VOSK and PyAudio, while text-to-speech feedback is provided using pyttsx3. SQLite is used to securely store user authentication data. The system supports execution of C, C++, Python, and Java programs using respective compilers and interpreters, and execution output is displayed in real time.

VI. RESULTS AND DISCUSSION

The system was evaluated using various voice commands across multiple programming languages. The results indicate that the assistant accurately recognizes commands and executes programs in offline mode. The integrated explanation feature improves conceptual understanding, and the system significantly reduces typing effort. Recognition accuracy may vary depending on background noise and pronunciation, but overall performance is satisfactory for academic use.The following figures show the results obtained from the implementation of the proposed AI-Based Voice Code Assistant.

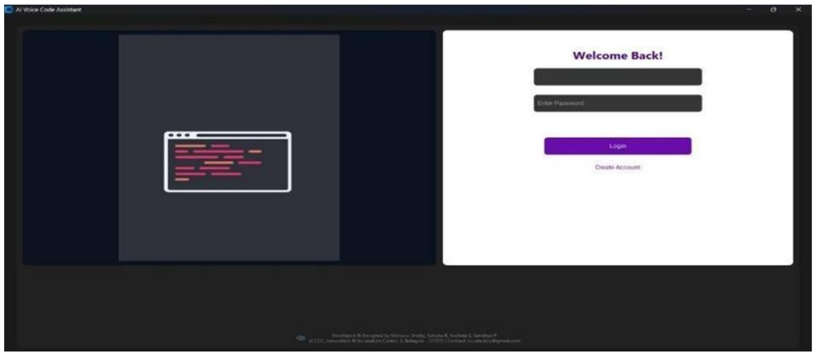


Fig. 4 User Dashboard Screen

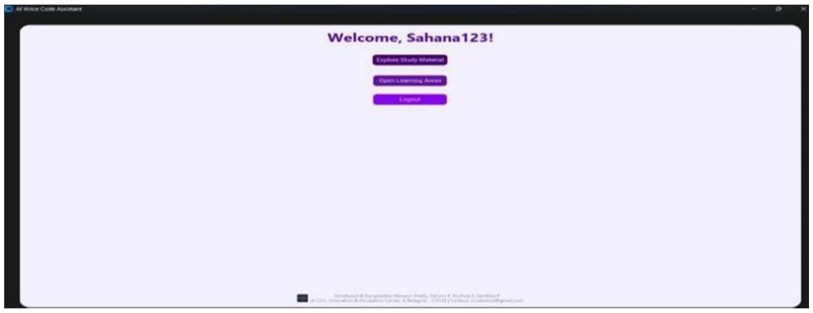


Fig. 5 User Dashboard Screen

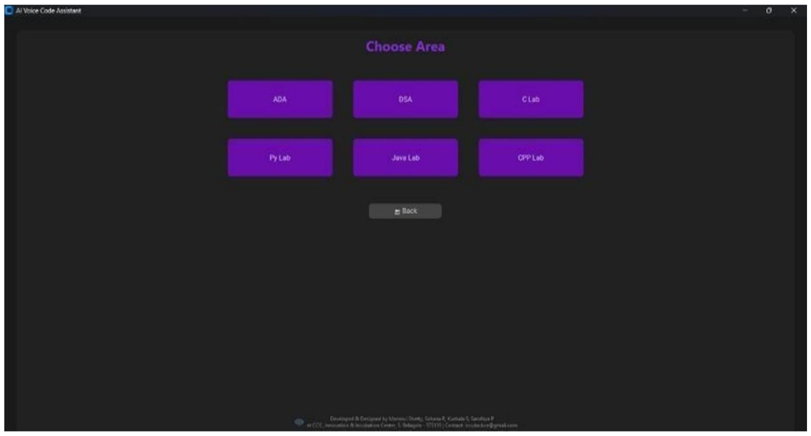
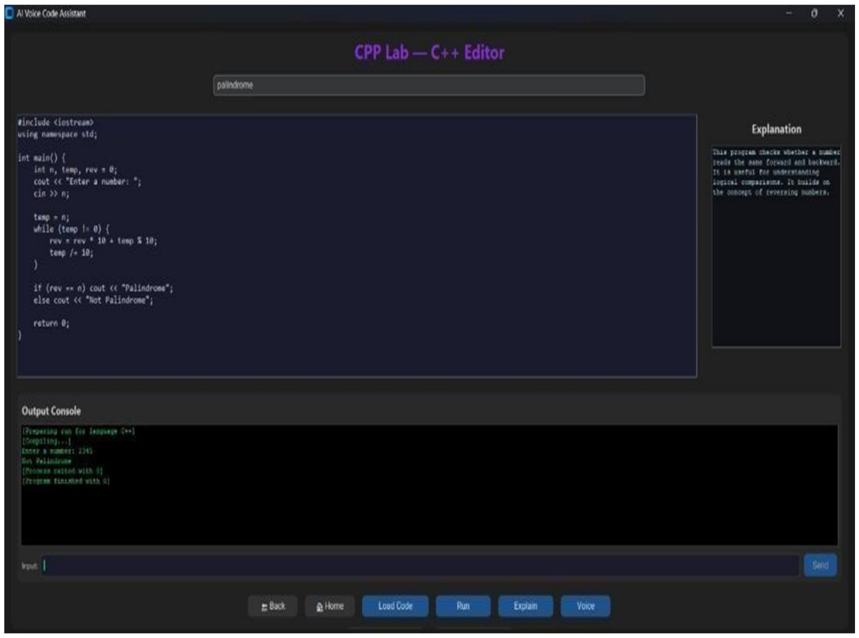


Fig. 6 Area Selection Page



```
#include <iostream>
using namespace std;

int main() {
    int n, temp, rev = 0;
    cout << "Enter a number: ";
    cin >> n;

    temp = n;
    while (temp != 0) {
        rev = rev * 10 + temp % 10;
        temp /= 10;
    }

    if (rev == n) cout << "Palindrome";
    else cout << "Not Palindrome";

    return 0;
}
```

This program checks whether a number reads the same forward and backward. It is useful for understanding logical comparisons. It builds on the concept of reversing numbers.

```
[Preparing run for language C++]
[Compiling...]
Enter a number: 2345
Not Palindrome
[Process exited with 0]
[Program finished with 0]
```

Fig. 7 Editor Interface and Execution of Program in Editor

VII. CONCLUSION

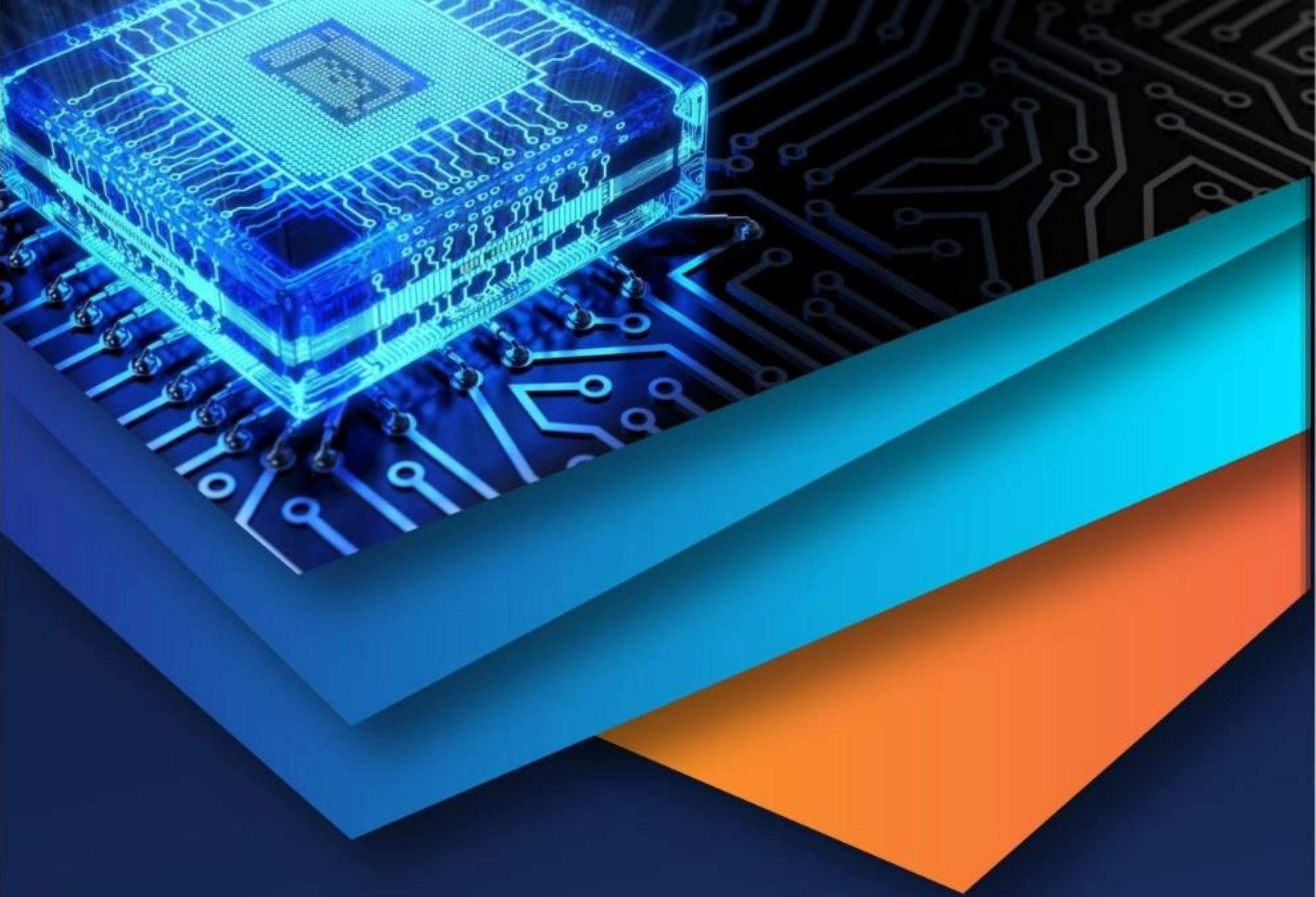
The AI-Based Voice Code Assistant demonstrates the effectiveness of voice-driven programming in offline environments. By integrating speech recognition, program execution, and explanation features into a single platform, the system enhances accessibility and learning efficiency. The proposed solution is well suited for academic laboratories and beginner programmers and provides a strong foundation for future enhancements such as expanded language support and adaptive learning features.

VIII. ACKNOWLEDGMENT

The authors express their sincere gratitude to the project guide and faculty members of the Department of Information Science and Engineering for their valuable guidance and support throughout the project.

REFERENCES

[1] R. Wataketiya, H. Malwatta, N. Chandrasiri, M. Nadeeshani, R. Kithsiri, and S. Siriwardana, “Voice Enabled Intelligent Programming Assistant,” International Journal of Computer Applications, 2022.
[2] A. Wagner, R. Rudraraju, S. Datla, A. Banerjee, M. Sudame, and J. Gray, “Programming by Voice: A Hands-Free Approach for Motorically Challenged Children,” in Proc. ACM SIGCHI Conf. Human Factors in Computing Systems (CHI), Austin, TX, USA, 2012.
[3] A. Désilets, D. C. Fox, and S. Norton, “VoiceCode: An Innovative Speech Interface for Programming-by-Voice,” in Proc. Int. ACM SIGACCESS Conf. Computers and Accessibility, 2006.
[4] T. B. Devshatwar, P. B. Deore, S. N. Awale, R. Jethure, and C. Nayak, “Voice Coding Using Artificial Intelligence,” International Journal of Engineering Research and Technology, vol. 10, no. 6, 2021.
[5] M. B. Garcia, J. B. R. Enriquez, R. T. Adao, and A. Happonen, “Hey IDE, Display Hello World: Integrating a Voice Coding Approach in Hands-on Programming Activities,” Education and Information Technologies, 2022.
[6] T. Zan and Z. Hu, “VoiceJava: A Syntax-Directed Voice Programming Language for Java,” IEEE Access, vol. 11, 2023.
[7] L. Rosenblatt, “VocalIDE: An IDE for Programming via Speech Recognition,” Bachelor’s Thesis, Carnegie Mellon University, 2017.
[8] A. Désilets, “VoiceGrip: A Tool for Programming-by-Voice,” Proc. Int. ACM SIGACCESS Conf. Computers and Accessibility, 2001.
[9] S. Nowrin, P. Ordóñez, and K. Vertanen, “Exploring Motor-Impaired Programmers’ Use of Speech Recognition,” ACM Transactions on Accessible Computing, 2022.
[10] P. Joshi and D. Bein, “Audible Code: A Voice-Enabled Programming Extension for Visual Studio Code,” International Journal of Computer Applications, 2020.



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)