Voice Activated Code Explainer for Visually Impaired Users

Priyanka S. Chavan[1], Harshal H. Abhyankar[2], Dhanashri S. Langade[3], Vyankatesh D. Gadekar[4], Ms. Swati More[5]

Department of Computer Science and Engineering,
D. Y. Patil Education Society's,
D. Y. Patil Technical Campus,
Faculty of Engineering & Faculty of Management, Talsande, Kolhapur, India.

***Abstract*—Programming has become an essential skill in the modern digital world; however, it remains highly inaccessible to individuals with visual impairments due to its strong dependence on graphical user interfaces and visually oriented code editors. Traditional Integrated Development Environments (IDEs) require developers to interact with syntax-highlighted text, menus, and complex visual layouts, which pose significant challenges for visually impaired users who rely on assistive technologies such as screen readers. These limitations create a major barrier to participation in software development and computer science education.**

This research proposes Voice-Input Code Explainer, an innovative web-based assistive programming platform designed to make coding more accessible through voice interaction. The system enables users to generate, understand, and execute Python code using natural language voice commands instead of manual typing. The platform integrates Web Speech API for real-time speech-to-text transcription and utilizes the Llama-3.3-70b-versatile large language model through the Groq API to generate structured Python code along with human-readable explanations.

The backend architecture is implemented using the Flask web framework, which processes user requests, interacts with the AI model, and ensures safe code execution through controlled environments and security checks. The system provides both code generation and detailed explanations of the generated logic, allowing beginners and visually impaired learners to understand programming concepts more effectively.

***Index Terms*—Assistive Technology, Voice-Input Programming, Web Speech API, Large Language Models, Python Code Generation, Digital Inclusion, Software Accessibility, Flask Web Framework, Natural Language Processing, Speech Recognition Systems, AI-Assisted Programming**

I. INTRODUCTION

In the modern digital era, programming has become a fundamental skill required in many fields such as software development, data science, artificial intelligence, and web technologies. Learning programming enables individuals to develop applications, automate tasks, and solve complex problems using computational logic. However, despite the rapid growth of the technology sector, programming environments remain largely inaccessible to individuals with visual impairments. Most programming tools and Integrated Development Environments (IDEs) rely heavily on graphical user interfaces, syntax highlighting, code indentation, and visual debugging tools. These visual elements create significant barriers for visually impaired users who depend on assistive technologies such as screen readers and voice navigation systems.

Visually impaired individuals often face multiple challenges while learning or practicing programming. Traditional coding requires constant interaction with text editors, keyboards, and visual cues, which makes it difficult for users who cannot easily interpret screen content. While screen readers can read out code lines, they may struggle to convey the logical structure, indentation, or syntax errors in a clear and intuitive way. As a result, many visually impaired students experience difficulties in understanding programming concepts, writing code efficiently, and debugging errors. These limitations create a digital divide where talented individuals are unable to fully

participate in software development simply because the tools are not designed with accessibility in mind.

In recent years, advancements in Artificial Intelligence (AI), Natural Language Processing (NLP), and speech recognition technologies have created new opportunities to improve accessibility in computing environments. Voice-based interfaces allow users to interact with systems using natural language instead of traditional input methods such as keyboards and mice. At the same time, Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding human language, generating programming code, and explaining complex concepts in a simplified manner. These technologies provide a strong foundation for building assistive systems that can help visually impaired users learn and practice programming more effectively.

The integration of speech recognition systems with AI-powered code generation models can significantly transform the way programming is taught and practiced. Voice-driven systems allow users to describe programming tasks verbally, while AI models interpret these instructions and generate corresponding code. Furthermore, AI models can provide step-by-step explanations of generated code, which helps users understand the logic and structure behind the program. This combination of voice interaction and AI assistance can reduce the dependency on visual coding environments and make programming more inclusive.

To address these challenges, this research proposes a web-based assistive programming platform called "Voice-Input Code Explainer." The main objective of this system is to enable visually impaired users to generate, understand, and execute Python code through voice commands. Instead of manually writing code, users can speak programming instructions in natural language. The system captures the spoken input using the Web Speech API, converts it into text, and sends it to a backend server built using the Flask framework. The backend then communicates with a powerful Large Language Model (Llama-3.3- 70bversatile) through the Groq API to generate Python code along with a detailed explanation of how the code works.

The generated output is displayed in the user interface, where the user can review both the code and its explanation. This feature not only helps visually impaired users create programs but also assists beginners in understanding programming logic. Additionally, the system includes a secure execution module that allows generated code to run in a controlled environment after performing safety checks to prevent potentially harmful operations. This ensures that users can experiment with programming safely without risking system security.

Another important aspect of the proposed system is its educational value. Many beginners struggle to understand programming concepts because traditional learning resources focus heavily on syntax rather than logical understanding. The Voice-Input Code Explainer addresses this problem by providing explanations alongside generated code, effectively functioning as an AI-based tutor. This approach helps learners grasp programming concepts more quickly and promotes interactive learning.

Moreover, the system supports an iterative development process, where users can refine or modify generated code through additional voice commands. For example, a user may initially request a simple program and later ask the system to add new functionality or optimize the existing code. This conversational interaction allows users to gradually build more complex programs without requiring deep prior knowledge of programming syntax.

The development of accessible programming tools is essential for promoting digital inclusion and equal opportunities in technology education. By enabling voice-based coding and AI-assisted explanations, the proposed system aims to reduce barriers faced by visually impaired developers and create a more inclusive learning environment. The Voice-Input Code Explainer demonstrates how modern technologies such as speech recognition, natural language processing, and large language models can be combined to build intelligent assistive tools that empower users with disabilities.

In summary, the proposed system seeks to transform traditional programming environments into a more accessible and user-friendly platform by leveraging voice interaction and artificial intelligence. By enabling users to generate, understand, and execute code through natural language commands, the system contributes to the broader goal of making programming education accessible to everyone, regardless of physical limitations.

II. PROPOSED SYSTEM:

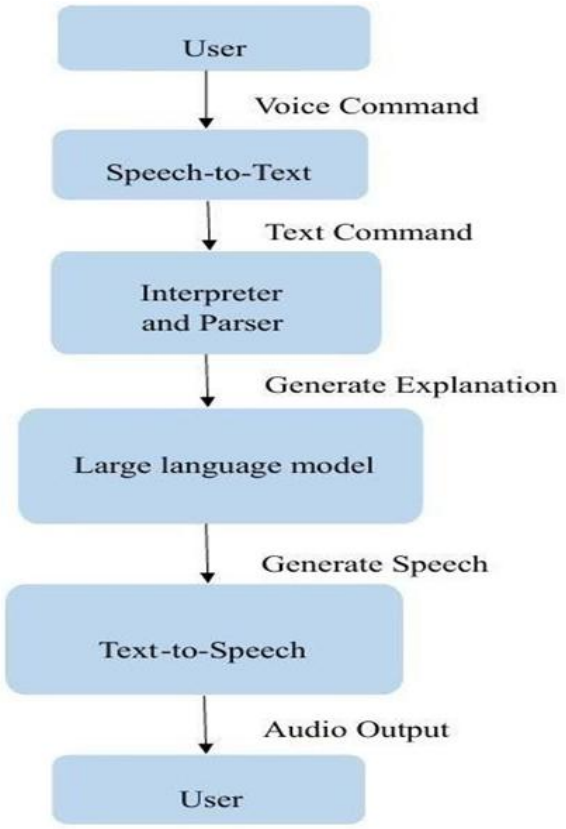


Fig. System Architecture

The proposed system, Voice-Input Code Explainer, is a web-based assistive programming environment designed to reduce accessibility barriers faced by visually impaired individuals in software development. Traditional programming environments rely heavily on visual interaction, keyboard input, and manual code editing. These requirements make it difficult for visually impaired users to navigate complex code structures or understand programming syntax efficiently.

To overcome these limitations, the proposed system adopts a Voice-First Architecture, allowing users to interact with the programming environment through natural language voice commands. Instead of writing code manually, users can describe programming tasks verbally, and the system automatically converts these instructions into structured Python code.

The system consists of three primary layers:

1. User Interface Layer

The frontend interface is designed using a high-contrast and accessibility-friendly layout to ensure usability for visually impaired users. It includes a voice recording trigger, command display area, generated code output, and explanation panel.

2. Processing Layer

This layer processes the transcribed voice commands and interprets user intent. The Flask backend receives user input and determines whether the request involves generating new code, modifying existing code, or explaining previously generated logic.

3. Artificial Intelligence Layer

The AI layer integrates the Llama-3.3-70b-versatile model via the Groq API. This model analyzes user commands and generates Python code along with detailed explanations that describe how the code works.

Additionally, the system allows users to update or refine previously generated code snippets through follow-up voice commands. This iterative interaction enables users to build programs step-by-step using conversational instructions.

By combining speech recognition technology with large language models, the system creates an intelligent voice-driven coding environment that simplifies programming and promotes accessibility.

III. LITERATURE REVIEW

1. Overview of Assistive Technologies for Visually Impaired Users

Assistive technologies have significantly evolved to support individuals with visual impairments in accessing digital systems. These technologies include screen readers, speech recognition systems, and text-to-speech engines that enable users to interact with computers using non-visual modalities. Recent studies highlight that speech-based interfaces allow visually impaired users to perform tasks efficiently by converting voice commands into system actions and responses.

ResearchGate +1

Voice assistants and conversational interfaces have further improved accessibility by enabling natural language interaction. Research shows that users with disabilities can effectively interact with such systems when they are designed with appropriate accessibility

guidelines and cognitive considerations.

2. Voice-Based Programming and Coding Systems

Recent research has explored the use of voice commands for programming to reduce dependency on traditional input devices. A notable system, Code with Disability, demonstrates a voice-command-based integrated development environment (IDE) that allows users to write, edit, compile, and execute code entirely through speech input.

ScienceDirect

Such systems convert spoken instructions into programming syntax using predefined datasets and structured mappings. While these approaches show promising results, they are often limited to specific programming languages and require predefined command structures, which restrict flexibility and scalability.

Additionally, conversational programming interfaces have been proposed to enable users to create and edit documents or code using natural language. These systems improve usability but still face challenges in handling complex logic, debugging, and real-time interaction.

3. Limitations of Existing Systems

1) Despite advancements, existing voice-based and assistive programming systems suffer from several limitations:
2) Many systems require complex installation and configuration, making them difficult for non-technical users to adopt.
3) Voice-based coding systems often rely on predefined commands, reducing flexibility in natural language interaction.
4) Lack of real-time feedback and streaming results leads to higher latency and reduced user experience.
5) Most systems do not fully comply with accessibility standards such as WCAG, limiting their usability for visually impaired users.

These limitations highlight the need for more flexible, scalable, and accessible solutions.

4. Research Gap

From the existing literature, it is evident that while speech recognition and AI technologies have improved accessibility, there is still a significant gap in developing advanced programming environments tailored for visually impaired users.

Specifically, current systems lack high-performance, zero-latency streaming AI editors optimized for WCAG AAA accessibility standards. Existing tools either focus on accessibility or intelligent code generation, but rarely combine both in a unified, real-time, browser-based environment.

5. Motivation for the Proposed System

1) The limitations identified in existing research motivate the development of the proposed system, Voice-Input Code Explainer. The system aims to address these gaps by integrating: Voice-driven interaction for hands-free programming
2) Real-time streaming code generation using AI
3) Multi-language support for broader applicability
4) Browser-based architecture to eliminate installation complexity
5) Accessibility features such as ARIA support and auditory feedback

By combining these features, the proposed system seeks to provide a modern, scalable, and inclusive programming environment for visually impaired users.

IV. WORKING OF THE PROPOSED SYSTEM:

The proposed system operates through a structured workflow that connects the client-side interface with the backend server and the AI model. The entire process consists of several stages that work together to convert voice commands into executable code.

1. Voice Command Capture

The user begins interaction by activating the recording feature through a microphone button on the interface. The browser utilizes the Web Speech API, specifically the web kit Speech Recognition engine, to capture spoken input and convert it into textual data.

2. Speech-to-Text Conversion

Once the voice input is captured, the Web Speech API processes the audio signal and produces a textual transcription. This text represents the user's natural language command, such as: "Create a Python program that calculates factorial of a number."

3. Backend Request Processing
The transcribed command is sent to the Flask server via a RESTful API endpoint called /generate. The backend server evaluates the request and determines the appropriate action, which may include:
Generating new code
Modifying existing code
Explaining a programming concept

4. AI Model Interaction
The backend constructs a structured prompt and sends it to the Groq API, which hosts the Llama-3.3-70bversatile model. The prompt instructs the AI to return a response in strict JSON format containing two fields:
Generated Python code
Explanation of the code logic

5. Response Parsing
Once the AI response is received, the backend validates the output using regular expression (regex) parsing to ensure the response follows the required JSON structure. This prevents formatting errors and ensures the frontend can display the information correctly.

6. Display of Results
The system displays both the generated code and the explanation in the user interface. This dual output helps users not only obtain working code but also understand the logic behind it.

7. Secure Code Execution
If the user requests execution, the generated code is checked for unsafe operations such as system commands, file manipulation, or external process calls. After passing the security checks, the code runs in a sandboxed environment to prevent potential security risks.

V. METHODOLOGIES AND MODELS:

The proposed system combines modern machine learning technologies with web development frameworks to create an intelligent voice-driven coding environment.

1. Llama-3.3-70B-Versatile Model
This large language model is selected due to its strong reasoning capabilities and ability to generate structured programming code. The model understands natural language prompts and translates them into syntactically correct Python programs.

2. RESTful API Architecture
The system uses Flask to implement RESTful APIs that connect the frontend interface with the AI backend. This architecture ensures modular design and efficient communication between system components.

3. JSON-Structured Prompting
The AI model is instructed to produce outputs strictly in JSON format. This approach simplifies parsing and ensures that the frontend can reliably extract both the generated code and its explanation.

4. Security Filtering
To prevent malicious execution, the system implements pattern-based filtering to detect unsafe Python libraries or functions. This ensures a safe learning environment for users.

VI. ADVANTAGES OF THE PROPOSED SYSTEM:

1. Reduced Typing Dependency
Voice-based input eliminates the need for extensive keyboard usage, which is particularly beneficial for visually impaired users.

2. Enhanced Learning Experience
The system not only generates code but also explains it in simple language, acting as an AI tutor for beginners.

3. Secure Programming Environment
The sandboxed execution environment prevents harmful commands from being executed accidentally.

4. Faster Code Generation
Integration with the Gro q API enables low-latency responses, allowing users to receive generated code almost instantly.

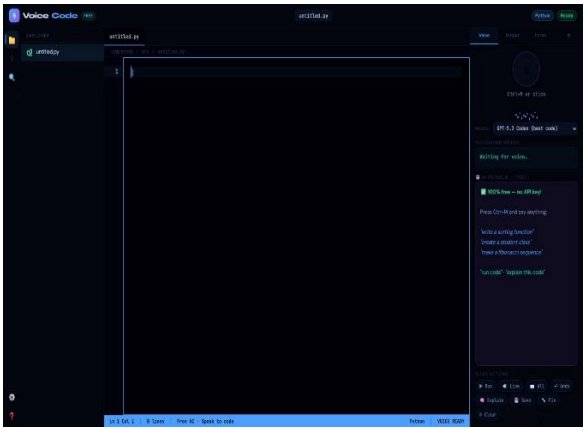
5. Improved Accessibility
The system makes programming more inclusive by

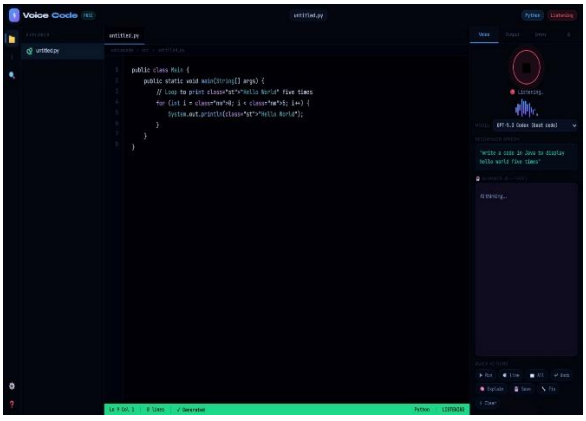
enabling visually impaired users to participate in software development.

6. Unified Platform

Unlike traditional tools that separate coding, explanation, and execution, this system integrates all three features into a single platform.

VII. OUTPUTS (SCREENSHOTS, GRAPHS, DATASETS):





VIII. FUTURE WORK

Although the proposed system provides an effective voice-based programming solution, several enhancements can further improve its functionality.

1. Text-to-Speech Integration

Adding a speech synthesis module will allow the system to read explanations and code outputs aloud, creating a fully voice - driven programming experience.

2. Wake-Word Activation

Future versions may implement wake-word detection, enabling users to activate the system using phrases such as “Hey Python”.

3. Mobile Application Development

Developing a mobile application will make the system accessible across smartphones and tablets

4. Voice-Guided Debugging

The system can be enhanced to explain runtime errors verbally and suggest corrections.

5. Multi-Language Programming Support

Future research may extend the system to support additional programming languages such as Java, C++, and JavaScript.

IX. CONCLUSIONS

The Voice-Input Code Explainer demonstrates how the integration of speech recognition technology and large language models can significantly improve accessibility in software development. By enabling users to generate, understand, and execute Python code using voice commands, the system reduces the dependency on traditional keyboard-based programming environments.

The platform provides both code generation and logical explanations, which makes it a valuable educational tool for beginners and visually impaired learners. Furthermore, the secure execution environment ensures safe experimentation with programming concepts.

Although the current system requires manual activation for voice recording, it establishes a strong

foundation for the development of a fully voice-controlled programming environment. Future improvements such as text-to-speech narration, mobile accessibility, and voice-guided debugging will further enhance the usability and impact of the platform.

Overall, the proposed system contributes to digital inclusion, accessible education, and AI-assisted programming, opening new opportunities for visually impaired individuals to participate in the field of software development.

REFERANCES:

[1] Gro q Cloud Documentation. (2026). Llama-3.3 Model Specifications and API Integration.
[2] Mozilla Developer Network (MDN). (2025). Web Speech API – Speech Recognition Interface.
[3] Pallets Projects. (2024). Flask Documentation: Secure Code Execution and Routing.
[4] Brown, T., et al. (2023). Large Language Models in Software Engineering. Journal of AI Research.
[5] Smith, J., & Patel, R. (2024). Assistive Technologies for Accessible Programming Education. IEEE Access.
[6] Johnson, L. (2023). Voice-Driven Interfaces in Human-Computer Interaction. ACM Computing Surveys.