Grenze International Journal of Engineering and Technology, June Issue

A Voice-Driven Intelligent Coding Assistant using Speech Recognition and Large Language Models

Sangita Gautam Lade[1], Payal Jannawar[2], Sanika Tapar[3], Rohan Somani[4] and Aryan Suratkar[5]
[1-5]Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, India
Email: sangita.lade@vit.edu, payaljannawar1@gmail.com, taparsanika@gmail.com, rohanjagdishsomani@gmail.com, aryansuratkar2005@gmail.com

***Abstract*— In this study, we are working to create a voice enabled intelligent coding assistant with the goal of simplifying and enhancing the efficiency of programming. Typically, in programming environments, programmers must do continuous typing and often move back and forth between different tools such as documentation, coding editors, and debugging environments which interrupts the programming process and ultimately decreases programmer productivity. The intent of the proposed coding assistant would be to allow programmers to create, modify or understand their programming code via voice commands thus increasing the accessibility of programming not only for professional coders but for students, newcomers, teachers, and others that would like to program with voice or have a hands-free computing experience. This coding assistant will provide the user with a speech recognition and natural language processing interface combined with large language models to convert voice instructions into executable code. The coding assistant uses a ReactJS web interface for the front-end development and has a FastAPI application for the back end and connects to the Google Gemini for processing the code. The MongoDB database will be used to store the conversational history, and the Monaco Code Editor will be utilized as an interactive development environment (IDE) for displaying and modifying the generated code.**

***Index Term*— Voice Programming, Speech Recognition, Natural Language Processing, Code Generation, Large Language Models, Human–Computer Interaction, Conversational AI, FastAPI , ReactJS, MongoDB.**

I. INTRODUCTION

Over the last few years, there have been considerable advancements in programming environments due to advances in Artificial Intelligence (AI), Natural Language Processing (NLP), and new designs for human–computer interaction. Despite these advancements in tools, many programmers continue to write code primarily through typing and switching among various tools, including, but not limited to, help files, source code editors, and debugging tools.

This constant switching can disrupt the flow of thought during code development and negatively impact a programmer's productivity. Furthermore, for beginner / novice programmers as well as for students and for developers who work in environments where it is not feasible to continuously use the keyboard, programming can be more difficult than for experienced programmers. Because of this, researchers are working on finding new methods for creating more measure space methods of programming.

Grenze ID: 01.GIJET.12.2.244_24

One avenue for exploration is voice-based programming. Through the use of speech recognition paired with large language models (LLMs), programmers will have the ability to verbally describe what they want the system to do in natural language. In return, the system could provide them with code suggestions and/or explanations for some portion of the task described. Voice-based programming creates an opportunity to have more interactivity during coding, which has the potential to result in less effort expended on repetitive programming tasks. This project develops a voice-assisted coding tool that enables users to use spoken commands when programming. The frontend of the coding assistant uses a ReactJS application for receiving input from users, while the backend uses FastAPI to accept and handle requests. The Google Gemini model is used for code creation, and MongoDB keeps track of the historical log of the user's past conversations with the coding assistant. The Monaco Code Editor provides users a place to see the newly generated code and modify it. This research aims to provide information about how combining voice input and AI generated coding may lead to improved flexibility in the programming process.

II. Literature Review

As a result the rise of artificial intelligence has made voice based coding increasingly common using advanced speech recognition technologies and natural language processing techniques. Various alternatives have been researched as developers can now communicate with programming systems using voice commands rather than through traditional keyboard functionality [8]. Specific studies have focused on helping individuals with disabilities by providing improved access to using programming languages, reducing the overall amount of typing required when writing code using a keyboard, and allowing programmers to use a natural means of interacting with their programming environments.

Begel and colleagues were among the first to investigate using speech recognition to program [7]. This research showed that programming interfaces can reduce the amount of excess typing done by programmers, but also support software developers experiencing physical stress from long periods of coding. While providing valuable ideas about the types of tools available, and suggestions for future uses of speech input programming methods, this and other studies indicate there remain several major challenges, including, the ability to accurately interpret (recognize) all of the different programming syntax, symbol and indentation based on user provided audio inputs.

To create systems that enable users to program by means of voice command (i.e., to convert a spoken instruction into computer program code) Hossain et al. developed a "voice command language system" that provides a set of standardised "voice commands" to enable the generation of "programming instructions" from "speech recognition" [5]. As this represented an improvement over traditional methods by reducing the need to memorize syntax, the system was based on "fixed command structures" (i.e., predetermined ways of issuing programming commands) and therefore did not have the same level of flexibility as when using "natural language" commands to issue programming commands.

Subsequent developments in this area have led to the creation of "conversational programming assistants" that use a combination of both "speech recognition" and "natural language processing techniques." Wataketiya et al. developed a "conversational coder assistant" [6]. This is a system that can interpret the user's intent from spoken instructions to provide code snippets in response. Although the conversational coding assistant is intended to improve user interaction with programming tools, the use of these types of systems is tempered by the difficulties involved in maintaining context throughout long and/or multiple interactions.

Data on AI systems that can generate automated code have recently been collected and analyzed through studies conducted to investigate LLM-generated code production. Zhao et al. outlined how they developed an interactive, self-generating code assistant using LLMs to generate code snippets and also improve on them using feedback mechanisms during the coding process [2]. They found that using LLMs to generate code provides improved results, while at the same time assisting programmers/code developers with programming tasks. Challenges still exist when generating incorrect or "hallucinatory" output from these algorithms.

Furthermore, Gupta et al. developed an AI-based voice executing verbalised commands and provides the ability to communicate verbally (speaking) through speech and understand/ interpret spoken (natural) language via speech recognition tools [3]. They also found high levels of accuracy when performing both functions; however, their system has challenges related to background noise interference when operating through the internet or wireless-based devices.

In addition to the previous studies, it appears that most current systems are either based on fixed vocal commands or can't maintain a consistent "conversation" over more than one voice interaction. Thus, there exists

an even greater need for a flexible voice coding assistant that both recognizes your spoken command & works great with AI -to interactively help you write code

III. METHODOLOGY

A. System Architecture

With the voice-driven coding assistant proposed (VDC), developers will be able to generate and interact with code through voice commands. The system's three-tiered architecture is composed of: a frontend layer, a backend processing layer, and a data and intelligence layer.

The architectural representation of the overall system is illustrated in Fig. 1. A user will complete an act using a given interface via the use of their voice commands. The user's voice command is an audio signal to the user's web browser, using the Web Speech Recognition API to translate the voice signal into text.

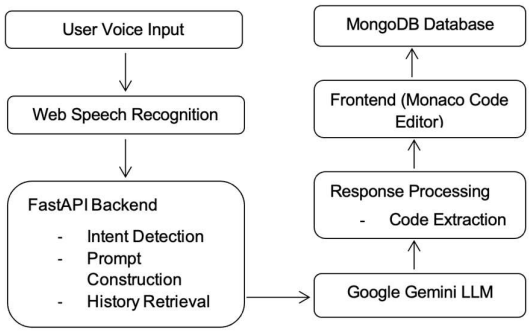

Fig. 1. System Architecture of the Proposed Voice-Driven Coding Assistant

After the voice command has been converted to text, the recognized text will be delivered to a backend application server for additional processing activities. The primary function of the FastAPI backend is to be the control center of the entire system; it takes the user's transcript, determines intent, retrieves previous conversation stored in the MongoDB database, and creates a structured prompt for the Google Gemini large language model.

From here, the prompt informs the language model and produces the appropriate code or explanation. When the response is returned, the response processing module extracts code blocks and removes unnecessary formatting or explanation. The resulting code is given to the user for viewing and editing through the Monaco Editor.

Additionally, the assistant stores all interactions in the MongoDB database to maintain context in multi-turn conversations and improve the quality of responses.

B. Frontend Layer

Using React with Vite as a Framework allows the creation of an interactive fast User Interface that will provide users with the ability to use a voice command to interact with the Coding Assistant and provide access to view real-time generated code.

The frontend has the following components:

VoiceControl Module

This sub-component of the Frontend captures the user's audio and converts it into text via the Web Speech Recognition API to provide real-time speech-to-text conversion.

CodeEditor Module

This is accomplished using the Monaco Editor library, which provides capabilities such as syntax highlighting, and formatting functions for writing code, like what you would see in modern IDEs.

Console Module

This displays responses from the system, as well as messages and error notifications generated by the assistant when code is being created.

UI Router

This manages navigation between various pages of an application: Home, Features, Documentation, Blog, and Main Code Interface.

All these components create an Interactive Environment to support Voice Driven Programming.

C. System Workflow

The workflow proposed in this document can be seen in Fig. 2. below.

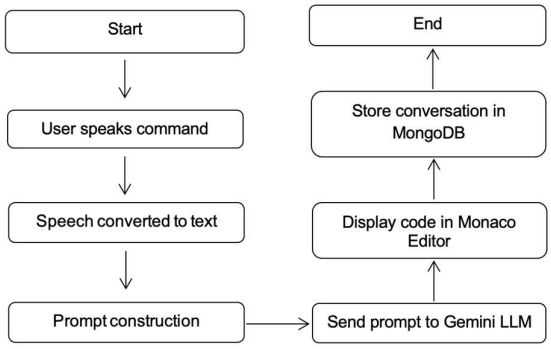

Fig. 2. Workflow of the Proposed Voice-Driven Coding Assistant

D. Backend Processing Layer

The backend of an application consists of FastAPI, which is responsible for processing requests received from the front end or client application. FastAPI facilitates communication between a user interface (presentation layer), a database, and the language model, and assists with controlling the actions within the application. It ensures smooth coordination between system components and manages the overall flow of data required for processing user commands efficiently.

The following steps constitute functions of the backend: receipt of the speech-to-text transcript from the client side and determination of the user's intended outcome by processing the content of the speech transcript to ascertain whether the desired outcome is code generation, code modification, code explanation, or a switch to another programming language. It also includes determining which programming language was requested in the command to create the best prompt for sending to the language model, forwarding the prompt to the Google Gemini model, processing the response from the language model, and saving all user interactions and responses provided by the AI assistant into an appropriate MongoDB collection. All these processes occur in a sequential order for the user to receive a command seamlessly without interruption to the communication process.

E. Data and Intelligence Layer

The Data and Intelligence Layer includes two components: Google Gemini large language model and MongoDB. Google Gemini model is responsible for generating code as its main function. Gemini uses structured queries from the backend to generate code or explanatory text in response to user requests.

MongoDB database provides persistent data storage for chat sessions. Stored in this database are user input, assistant output, timestamps, and other chat metadata. Having this information allows the system to maintain context throughout the conversation and support multiple turns of interaction.

IV. IMPLEMENTATION

This Voice-Enabling I.C.A. (Intelligent Coding Assistant) is a Fully Integrated Web Application built with React frontend, FastAPI backend, MongoDB database, and Google Gemini LLM for code generation and explanation. Supports Real-Time Voice Interaction, Contextual Conversation Retention, and Creation of Multi-Program Language Code.

A. Frontend Implementation

The user interface for this application created with the React framework has been developed using Vite as the build tool, creating quick real-time interaction between the user and the application. The application processes voice commands through Web Speech Recognition API using the in-browser speech recognition capabilities, which generates output in the form of transcribed text. This provides low latency processing of voice commands without the need to create a separate server. The User Interface features a Monaco Editor which allows developers to work on the generated programs in a manner similar to an Integrated Development Environment. The Monaco Editor supports many programming languages and provides features including syntax highlighting,

auto formatting and structured view of the programming code. Users can interact with the generated program codes using an IDE-like method. Refer to Fig. 3. for the graphical representation of the proposed system's user interface.

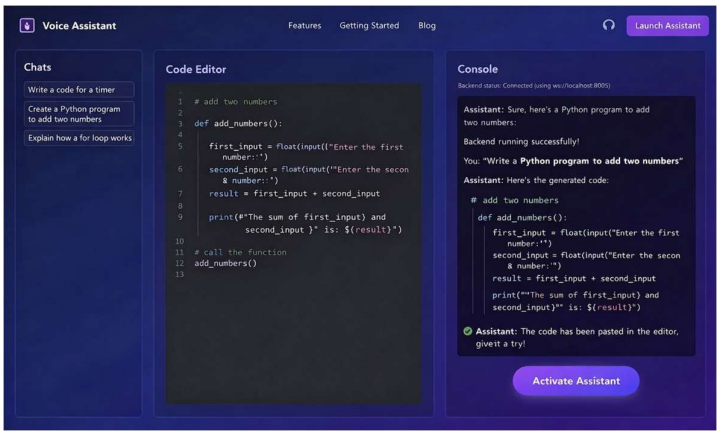

Fig. 3. Voice-Based Code Generation Interface Showing the Chat Panel, Monaco Editor, and Assistant Console.

A minimalist state management system is being implemented to keep track of when users interact with and create sessions interacting with the system. The basic structure of the application has been established using a react router for navigation through all sections of the application, including home, workspace, features, documentation, and blog.The front-end and back-end communicate through a WebSocket connection using Socket.io-client to maintain a persistent connection for real-time message exchange between the user interface and back-end server.

The user interface has been designed to be interactive to allow users access to the functions of the system:

- A Microphone button has been created that allows the system to receive voice commands from the user, and a Message Console has been created that will display all incoming data that is sent from the assistant.
- A Chatsidebar has been created to allow user to see a complete history of all previous chats with the assistant.
- A Suggested panel will be created and filled with the language model used to create the suggestions.

Finally, the application has been built and designed using Tailwind CSS; therefore, the application will look great and work well on all devices.

B. Backend Implementation

FastAPI is often a popular option when developing a back end that regularly connects with AI models since it can efficiently manage asynchronous requests while providing high-performance API services.

The back end comprises three REST API endpoints:

- /code-assistant assists the user in generating or explaining a piece of programming code
- /chat-message establishes and manages an interaction with a chat feature
- /chats gathers records of chat sessions and assists in managing the history of chat sessions

The back end is integrated with the Google Gemini 2.5 Flash model that takes in prompts from users via natural language promises back out programming language output or trigger instructions with associated explanation.

The processing layer supports the retrieval and understanding of user input through 3 primary functions:

- To provide a method of detecting user intent (e.g. generate code, modify code, or explain code)
- To detect words related to programming languages
- To provide summaries of prior chat interactions to give context in a capacity to be inferred with multiple prior interactions

This supports effective processing of user intent through the backend system and allows context to be accumulated through multiple interactions in addition to overall user intent.

C. Database Implementation

To store session and conversation histories, the system stores them in a database called MongoDB. The use of MongoDB as a document-based Data Storage model was selected because it supports storing data related to conversations.

The two different types of collections that exist for storing this data are:

- Chats Collection - contains chat session-related metadata such as Session title, timestamps and user Id.
- Messages Collection - contains individual messages exchanged between the User and Assistant such as the Message Content, Role ID and Timestamp.

Storing data in these two types of collections allows for the fast retrieval of past interactions with an Assistant when a User is involved in the context of a multi-turn session.

D. Prompt Engineering and Context Summarization

As a means of improving response quality, this system creates a hybrid approach to prompt engineering methods and techniques for summarizing contextual information. This approach combines summarization of previous messages in conversation, analyzing total number of tokens utilised and enhancing messages created earlier in the conversation.

This allows the system to retain relevant contextual information while maintaining its maximum token limit. Another benefit is that these techniques aid in providing fallback mechanisms when the language model produces incomplete, erroneous, or no outputs, increasing system reliability and enabling stable system operations.

E. Deployment and Execution

The app uses several parts to execute.

- The frontend uses the Vite development server
- The backend uses the Uvicorn ASGI server with FastAPI for serving HTTP requests
- The database is run using a MongoDB instance either locally or in MongoDB Atlas
- The app uses the Google Generative AI SDK for their LLM Gemini integration

All sensitive configuration data, like API keys and database connection strings, are stored in the environment variables for secure deployment.

V. Results and Performance Analysis

A. Code Generation Accuracy

Several experimental processes were conducted utilizing multiple types of natural language prompts with the goal of assessing how effective the voice coding assistant could perform as a virtual tool. Testing with 5 programming languages (Python, JavaScript, Java, C++, & C#), over 80 individual tests were performed with each randomly generated output being manually examined for correctness.

86% of the randomly generated code outputs were deemed as correct, 11% required minor modifications, while the remaining 3% of the code outputs were not deemed as valid or included hallucinated code outputs. The results of this study support prior research on the use of large language models (LLMs) to create computer programs. Past research reported that LLMs created syntactically correct code with 80-90% accuracy; however, due to the presence of contextual information in the conversation and the use of an iterative method, these factors resulted in greater quality of output than LLMs alone.

The graphic provides distribution data on the numerical results associated with accuracy of code created from LLMs.

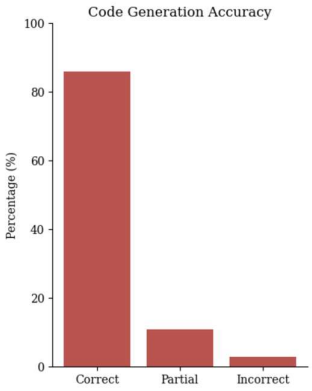

Fig. 4. Distribution of code generation accuracy for the proposed voice-driven coding assistant.

B. Speech Recognition Performance

The speech recognition engine built into the web browser provides an instantaneous means of converting voice commands to text. As a result, there is lower reliance upon external voice processing systems, and this implementation has been proven to reduce latency.

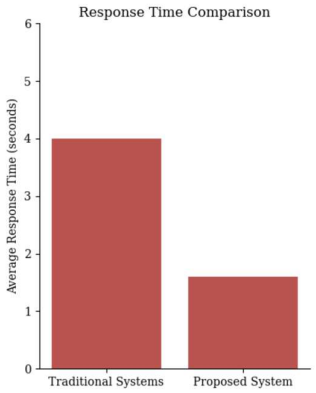

Fig. 5. Response time comparison between traditional voice coding systems and the proposed system.

Based on experiments conducted, the speech recognition module provided 92–95% accuracy while converting voice to text during general conversation-style programming. The reliability of this system has also proved reliable when using voice commands related specifically to coding and explanation purposes.

As per response time from when a user issues a command with their voice until code is outputted, it took between 1.2 seconds and 2.0 seconds to get an output response. This is much faster than the average of 3 to 5 seconds it takes other commonly used voice-driven programming systems.

A side-by-side comparison was made of the response times measured using traditional voice-driven programming systems compared to the proposed system shown in Fig. 5.

C. Intent and Language Detection

A lightweight module for inference exists in the system to determine the user's request for programming using voice commands, including the desire for a specific programming language. The evaluation results show that intent was detected with 98% of the time completed for three tasks: creating code, explaining code, and modifying code. Also included in determining what the programming language intended to be used was through keyword context with 94% accuracy, which provides a reliable way to interpret user voice commands and allows for seamless communication while performing programming tasks using voice command.

D. User Experience Evaluation

In our initial usability assessment, 15 users were asked for their thoughts on the usability of the system as a whole process by rating individual elements using a 5-point Rating Scale. Each user interacted with the system through both voice and graphical means.

Below is a summary of the scores that were given in the initial assessment, by user:

- Ease of Use = 4.6/5
- Accuracy of Code Developed = 4.4/5
- Voice Interface Usability = 4.5/5
- Contextual Memory Efficacy = 4.7/5

Overall, most users indicated that the combination of voice commands, feedback, and the ability to visually see the code being developed via a graphical interface helped them greatly increase their productivity in developing software.

E. Comparative Analysis

The proposed programming assistant has been compared with other assistants in the market using criteria such as multi-language support, contextual memory, and code editor integration. Most assistants provide limited multi-language support and do not have contextual memory for multi-turn conversation.The proposed voice-based programming assistant offers multi language support and retains previous interaction history so that the user and the assistant can interactively work together to complete programming tasks effectively. The association with the

Monaco Editor improves how generated code is displayed and edited. The comparative analysis can be found in Table II.

TABLE I: COMPARISON OF VOICE-BASED PROGRAMMING SYSTEMS

Feature	Existing Systems	Proposed System
Multi-language Support	Limited	Extensive (Python, JavaScript, Java, C++, etc.)
Contextual Memory	Rare	Fully Implemented
Speech Recognition	Moderate latency	Near real-time
Code Editor Integration	Weak	Monaco-based editor
LLM Model	Varies	Gemini 2.5 Flash
Accessibility	Medium	High (voice-first design)

VI. CONCLUSION

An intelligent coding assistant that utilizes voice commands has been developed as described in this research project. Users can speak codes into the system and have them generate code. This intelligent coding assistant leverages speech recognition, natural language processing, and large language models together with a web architecture (React front end, FastAPI back end, MongoDB database). Results of experiment indicate high accuracy for code generation, strong performance in speech recognition, and low latency for responding. When a user interacts with the system through voice command, they are able to use contextual memory to complete code creation faster than traditional means. These research findings validate that voice-assisted coding tools increase accessibility and productivity of developers. Future directions include increasing code generation reliability and providing support for languages.

FUTURE WORK

Future improvements will include integrating advanced large language models with effective prompt optimization techniques so the generated code has greater accuracy and reliability. It may also add additional programming languages and software development frameworks.The system will also support offline speech recognition, better recognize a speaker's intent when generating output, and have more direct connectivity to Integrated Development Environments. The next version may employ contextual learning to understand longer programming-related conversations and provide better responses to the user. With these improvements, voice-command programming assistance systems are expected to become more useful and effective as programming assistants.

REFERENCES

[1] T. Zan and Z. Hu, “VoiceJava: A syntax-directed voice programming language for Java,” Electronics, vol. 12, no. 1, p. 250, 2023.

[2] Z. Zhao, J. Sun, C.-H. Cai, and Z. Wei, “Code generation using self-interactive assistant,” in Proc. IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC), 2024.

[3] S. Gupta, M. Haider, and M. Shabbir, “Development of an AI-powered voice assistant: Enhancing speech recognition and user interaction,” International Journal of Scientific Research in Science, Engineering and Technology, vol. 12, no. 3, pp. 717–727, 2025.

[4] S. McDaniel and M. F. Zibran, “Improving source code with assistance from AI: A pilot case study with ChatGPT,” in Proc. 7th International Conference on Information and Computer Technologies (ICICT), 2024.

[5] A. R. M. Nizzad and S. Thelijjagoda, “Designing of a voice based programming IDE for source code generation: A machine learning approach,” in Proc. International Research Conference on Smart Computing and Systems Engineering, 2022.

[6] S. Mukherjee, A. Nediyanchath, A. Singh, V. Prasan, D. V. Gogoi, and S. P. S. Parmar, “Intent classification from code mixed input for virtual assistants,” in Proc. IEEE 15th International Conference on Semantic Computing (ICSC), 2021.

[7] M. Desilets, “Programming by voice: A domain-specific application of speech recognition,” in Proc. International Conference on Intelligent User Interfaces, 2018.

[8] R. Kumar and S. Singh, “Voice-enabled intelligent programming assistant for automated code generation,” World Journal of Research and Review, vol. 13, no. 5, 2021.

[9] A. Sharma and P. Verma, “Voice-controlled smart programming assistant using speech recognition,” International Journal of Advanced Computer Science and Applications, 2023.