iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VII **Month of publication:** July 2026

DOI: https://doi.org/10.22214/ijraset.2026.84182

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 14 Issue VII July 2026- Available at www.ijraset.com

CodeByVoice: A Voice-Controlled Python Programming Assistant for Differently-Abled Users

Epili Santosh Kumar Patro[1], D. Shiva Naik2
[1]*M.tech Student, CSE DEPT, Lenora College of Engineering, Rampachdavaram, AP*
[2]*Assistant professor, Lenora College of Engineering, Rampachdavaram, AP*

Abstract: ***CodeByVoice is a programming assistant which you control with your voice; it helps you code in Python without having to touch a keyboard via natural language commands. We have a mic which captures your voice and a speech recognition engine which turns that into text. That input is then processed to produce correct Python code as per your instructions. CodeByVoice includes a range of features from generating new code, to editing what you already have, running programs and saving files with simple voice commands. Also, we have a graphical interface which displays the code which is generated, you may review it there or go in and make manual changes if you wish. By running the program, the program will capture the output to display it and have the results read aloud via a text-to-speech engine. The program will also manage files by saving the user-created programs as Python (.py) files by name. The Error handling Mechanism improves seamless interaction by recognizing user input as invalid commands or a violation of the program's syntax and offers guidance to remedy the issue. CodeByVoice offers the most streamlined and user-friendly environment for programming by offering the ability to code and execute code using speech, as well as receive audio feedback. This is also very useful for novice users, users who code for utility and those who prefer not to use a keyboard for coding.***

Keywords: Voice-controlled programming, speech recognition, natural language processing, python code generation, assistive programming technology, human-computer interaction, accessibility.

I. INTRODUCTION

In today's world, programming is an essential skill. Research being done in this area backs the present innovations in the world. Programming is being used in applications based on Artificial intelligence, data science, software development, etc. A Brief Look at Programming Environments. Integrated programming environments did not develop overnight. The focus of integrated programming environment is language and tools. Besides, productivity tools are being used by programmers too. A system that includes an automated debugger, IDE and testing system. Furthermore, a built-in debugger module is also created for cross-language debugging. The code combine system is designed to automatically accept, reject and coordinate patches now. Also, code combination is incorporated into the coding environment. We also develop a cognitive model that predicts a programmer's eye movement. Moreover, the IDE created for cleaning code is IntelliJ. A code assistant has been structured to help the planning process. An error detection and prevention mechanism are included. Also, the code assistant incorporates the planner process.

Moreover, the pre-planning, Advances in speech recognition and voice-based interaction are now offering interesting possibilities for tackling the issue. In recent times, voice user interfaces have gained immense traction and have found applications in multiple spheres. A researcher has studied if programming with the use of a voice is a possibility or not. Numerous methods for using voice interaction for programming have been proposed, but most of these approaches are either restricted to only dictation or very limited translation of natural language to programming language. Additionally, they lack real-time execution capability and cannot edit program output or read it aloud. Therefore, it is incorrect to say that one can program completely through voice. All the previous systems discussed in the literature review accept voice as input and convert it to corresponding text. Nevertheless, no system method defines the system that uses the text as a programming language and marks the steps of the programming language. The above approaches lack proper fault identification since they only deal with the rectifications in the statement. It is to be noted that in all the above systems author is accepting the voice input for only one statement at a time. The complexity of the statements that each system can handle is one issue common to all of the approaches. The complexity can be measured by the number of operators, characters, identifiers, keywords, etc. A programmer's assistant has high complexity which was less in case of earlier approaches.

II. LITERATURE REVIEW

The voice-based interaction has received a significant amount of scholarly interest in the field of human-computer interaction research, with the development of speech recognition and natural language processing increasingly becoming more developed. Speech-recognition technologies translate verbal communication into text that can be read by computers, allowing to perform the work with the help of voice commands and thus being the core of modern digital assistants and interactive programs. Early rule-based models Speech recognition has moved through the simpler models with early rule-based models, to more advanced deep-learning models able to work with large vocabularies with greater accuracy and robustness. Early work in this area showed that speech-based interfaces could offer other forms of interaction to users who had problems with the traditional input devices in which case the concept of coding proved to be highly relevant to the research as it is often necessary to spend a lot of time typing in a lot of code with the help of the normal keyboard. Other researchers explored the viability of voice programming and proposed the idea of spoken programs, which are programs expressed verbally by a developer and are structured using spoken commands and programming constructs [1]. The lexical and syntactic ambiguities of spoken code that were previously unknown to them before their work were therefore, to make their interpretation a complex task. Similarly, early speech recognition systems like CMU Sphinx demonstrated that continuous-speech recognition systems were capable of supporting large-vocabulary recognition and could be used as the foundation of voice-based applications; however, it was not specifically oriented to the programming environment, which needs accurate syntax and structure program commands.

The further studies went beyond the dictation-based coding methodology and delved into the concept of natural-language programming. Some of them explored the way programmers in their novice stages represent computational tasks in a natural language and contended that natural-language interfaces could make programming more user-friendly and their results indicate that users are more inclined to describe logical operations using their speech than by writing down precise syntax [2]. Based on this foundation, a person created Spoken Java which was a voice-friendly version of the Java programming language which allowed programmers to dictate code in a more natural manner. The system also included the use of parsing to overcome ambiguity in the spoken input and it proved that programmers could also learn to write code utilizing voice-recognition technologies. Other research explored conversational programming systems where the user communicates with the system through speech or text. Indicatively, later created a chat style programming interface and stated that novices were excited about voice-based coding but experienced programmers continued to prefer a voice-text interaction in code accuracy [3]. Nevertheless, with these developments, most of these systems were experimental and focused on specific parts of a programming process, like dictation or command execution, but not the entire development process.

Similar significant progress in speech-recognition studies has significantly improved the technical viability of voice-based programmers. Modern deep-learning models, such as neural-network-based systems, have lowered the error rate in speech-recognition and made real-time speech-recognition tasks accurate, by providing correct transcription. Experiments on end-to-end speech-recognition systems like Deep Speech showed that neural-network models had strong ability to learn effective acoustic representations and beat traditional systems in noisy environments. Similarly, attention-based sequence-to-sequence models have demonstrated better recognition in voice search and dictation applications, highlighting the potential of the more sophisticated neural networks to speech-based applications. Other speech-recognition engines like Julius engine have been extensively used in research to build real-time speech-recognition systems with the ability to support large vocabularies. Such advances in technology have promoted the creation of voice interfaces in different fields; however, the implementation of speech recognition on programming poses more issues since the programming languages have stringent syntax. Even a small amount of transcription errors may lead to unparody code, so sound command interpretation and error management are vital ingredients of voice-based programming systems.

Further studies in assistive technologies provide strong support to the need of accessible programming platforms. The motor impaired programmers also find it hard to type long durations of time, thus alternative methods of input like voice based code systems are developed. Research on the vocal programming environment, including VocalIDE [4], has shown that voice interfaces could be helpful in code navigation and editing tasks and minimize reliance on manual input interfaces. However, they usually depend on speech-recognition technologies currently available and cannot properly edit complicated programming constructions. Furthermore, most solutions are more focused on code entry and less integrated they tend to offer features like automatic code generation, program execution and audible output feedback. This division points out a main limitation of the available literature: most of the studies refer to the isolated parts of voice-based programming instead of developing an overall system that would facilitate the entire programming process.

In general, the literature unveils a number of critical trends and disjunctures. Previous studies have demonstrated the viability of speech-based programming and shown the potential to achieve benefits in terms of natural-language interface. The technical basis of such systems has been further made solid by spurred developments in speech-recognition and machine-learning. However, most of the current solutions are based on strict command hierarchies or on narrow functionalities, including dictation or navigation. Not many systems combine speech-recognition, natural language command interpretation, auto-code generation, program execution and voice feedback in a single environment that has been designed explicitly to be accessible. Furthermore, although the literature has encountered the issues associated with different abilities in programmers numerous times, only a small number of practical instruments have been designed with accessibility as a key design focus. This gap is filled by the current research by introducing CodeByVoice, a voice-activated Python programming assistant, which combines speech-recognition, command-interpretation, code-generation, code-execution and text-to-speech-feedback into one interface. Through these elements, the study will also be used to help create user-friendly and user-inclusive programming environments that enable users to write and control code using natural-language voice commands.

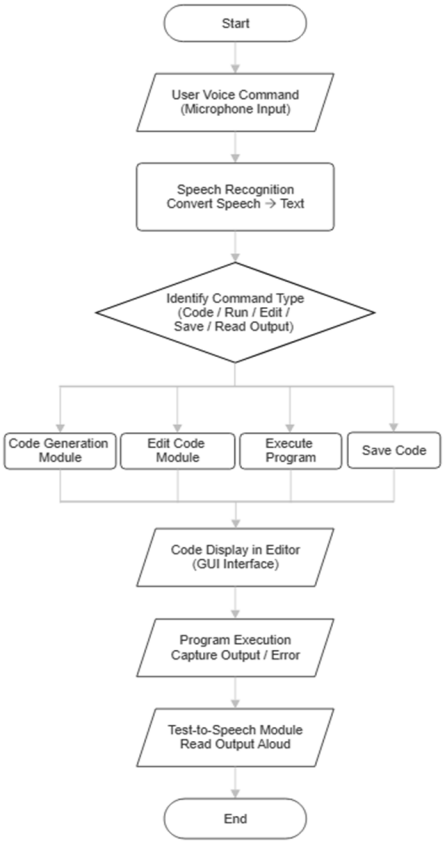
III. PROPOSED WORK

The research proposal will use a qualitative system design methodology to design and test a voice-controlled programming assistant called CodeByVoice. The main goal is to develop a prototype that allows running Python programs through natural-language voice commands instead of the conventional keyboard input and allowing them to be created, edited, and managed through it. The system is developed in the Python-based development tools and speech-processing libraries in a controlled laboratory setting. The study period is carried out on standard desktop systems since development and testing is done. A qualitative design will be suitable to use in the current work since the study does not only consider the technical implementation but also examines the potential of voice interaction to redefine the programming workflow and enhance accessibility to users who might have issues with traditional coding interfaces. These exploratory design methods are common in human-computer interaction studies in order to test new interaction paradigms and usability factors [5].

The proposed system architecture provides a number of functional modules that, when combined, can be used to support voice-driven programming. This starts with a speech input unit where the spoken commands are captured using a speech microphone. Through these audio cues, a speech recognition engine is used that translates speech into textual instructions. The current neural-network-based speech recognition has enhanced the accuracy of the continuous speech processing, and thus has enabled natural-language interaction to be practical in software programs [6]. A command interpretation module then analyzes the recognized text and determines the intent of the user and classifies them as either code generation, execution, editing or file manipulation. The processing of these instructions is done using natural-language processing methods which convert them into the structured programming activity that can be executed by the system.

After the command intent has been determined, an interpretation module is used to convert the interpreted instruction into Python code that is syntactically correct. This module uses rule-based logic and a set of templates to create generic constructs e.g. loops, conditional statements, functions and input/output operations. Python is the choice of implementation language because it is a readable language and is also very common in education and business. The resulting code is shown in a graphical code editor that is built on a lightweight framework like Streamlit, and the user can visually inspect and edit the program when needed. The use of voice interaction, as well as visual confirmation, enhances the usability because hybrid interfaces allow users to check and correct system outputs more efficiently [7].

The other modules included in the system are program execution, auditory feedback, and file management to enable full voice-based development workflow. Generated programs may be executed by voice command and once this happens the execution module runs the script and presents the output or error messages. A text-to-speech feature translates program output into audio feedback so that users can hear program outputs without necessarily having to use the display only. Besides, there is voice-based management of files where the user can store a program with command like save this program as factorial to save the generated code in a Python file under a defined directory. The proposed system will go beyond the simple voice dictation applications because it presents speech recognition, command interpretation, code generation, execution, and feedback controls in the same interface and will be designed to be more valuable and accessible to a broader audience of users, including differently-abled programmers.



IV. RESULTS AND DISCUSSIONS

In the proposed CodeByVoice system, the experimental demonstration and analysis indicates that voice-based interaction can be used as a practical interface to programming environments. The prototype was able to receive spoken input, translate voice commands into text, decipher the desired programming command, and produce syntactically correct Python code in response to some simple programming tasks. Moreover, the system ran the programs generated and provided visual and auditory feedback based on the text-to-speech module, which formed a full voice-based programming environment. Such results imply that implementing speech recognition, command interpretation, and automated code generation as a part of a single platform can have a significant impact in helping hands-free coding. Regarding accessibility, the system has an alternative interaction mechanism, which minimizes the use of keyboards and manual input devices. This is in line with general concepts of accessible computing and human-computer interaction, which are based on the need to engineer technologies that are inclusive of diverse user abilities and preferences in interaction [8]. In its turn, the findings confirm the main hypothesis of this study according to which voice interaction can be implemented in programming environments and it does not change the logical organization of the coding operations.

Looking at it in the context of the previous studies, a number of areas of agreement and further development can be identified. Previous experiments on voice programming, especially showed that speech-recognition systems were capable of dictating code, but that these systems required the user to speak specific syntax of code [9]. This made such requirements add to cognitive load and even constrained usability. Conversely, the current system processes natural-language command, and translates it into interpretable Python code, thus lessening the necessity of users to speak precise syntax formulations. This observation is also in line with those reported by who reported that inexperienced programmers generally tend to think about programme logic in the form of ordinary language instead of writing formal expressions of the code [10]. The present work further adds to the conceptual studies that have been made previously by operationalizing this idea with respect to natural-language command interpretation into a practical implementation. Furthermore, as opposed to the previous systems where the main emphasis was placed on a single separate functionality, e.g., voice-based code dictation or navigation, the suggested solution combines several elements, e.g., code generation, execution, editing, and file management, in a single interface. This integration marks a significant improvement in the direction of a more complete voice-based programmable environment.

Although these are promising findings, they are limited by a number of constraints which indicate that voice-based programming systems still face challenges. A major problem is associated with the accuracy and reliability of the speech recognition when used in the programming environment. Despite the fact that current neural-network-based speech recognition methods have been able to drastically enhance the performance, programming tasks require very high accuracy due to the fact that even a minor error of recognition can result in an invalid syntax or incorrect programme behavior [11]. In the testing process, certain complicated or unclear voice instructions led to the generation of incorrect codes, and the user had to repeat or restructure the voice instructions. It is not the first time the current observation is echoed by prior findings that conversational programming interfaces are more effective with visual confirmation and manual correction facilities [12]. Furthermore, the present prototype is concerned with the basic Python syntax (loops, conditionals, input-output procedures, etc.). Although it is enough to show that feasibility is possible, more complex programming situations that require complex libraries or complex logic would need more sophisticated natural-language understanding and situation-specific reasoning. These weaknesses suggest that future studies need to investigate better models of command-interpretation, bigger studies of usability involving more able programmers, and the support of more programming languages and development tasks. This might also enhance the capability of voice-based programming to develop more accommodating and more flexible programming environments.

V. CONCLUSION

The paper had suggested and created CodeByVoice, a voice-activated Python programming assistant, which allows users, especially people with disabilities to interact with programming environments through natural language voice interaction. Its inherent goals were enabling a dependable speech-recognition module, syntacticizing the spoken instructions into runnable Python code, running programs with both visual and audio feedback and allowing the user to edit, maintain and save code files using voice-interaction. Its results showed that the system was able to translate spoken commands into executable Python programs, allow simple program programming structures and display output in a graphical and text-to-speech format. The results of these experiments show that it is possible to combine speech recognition, command interpretation and code generation in one interface and develop a functional and convenient programming workflow that reduces the need to use traditional keyboard-based code writing habits.

In addition to the technical implementation, the study also advances the overall research on the accessible computing and human-computer interaction area by showing how accessible programming environments may be made as multimodal interfaces including voice input, visual code representation and auditory feedback. However, the research is vulnerable to a number of limitations. The existing prototype focuses on simple programmable constructions and is dependent on how precise the current speech-recognition systems are and can still give some errors to interpreting complex commands or diverse speech patterns. The system could be improved in the future by adding more sophisticated natural language processing features, expanded support of more advanced programming tasks and testing it with users who have a wide range of interface accessibility requirements. These developments would be supportive of the functional usability of voice-based programming software. On the whole, this study is just a first step to more inclusive programming tools and a better understanding of how voice-controlled technologies can alter software development is possible.

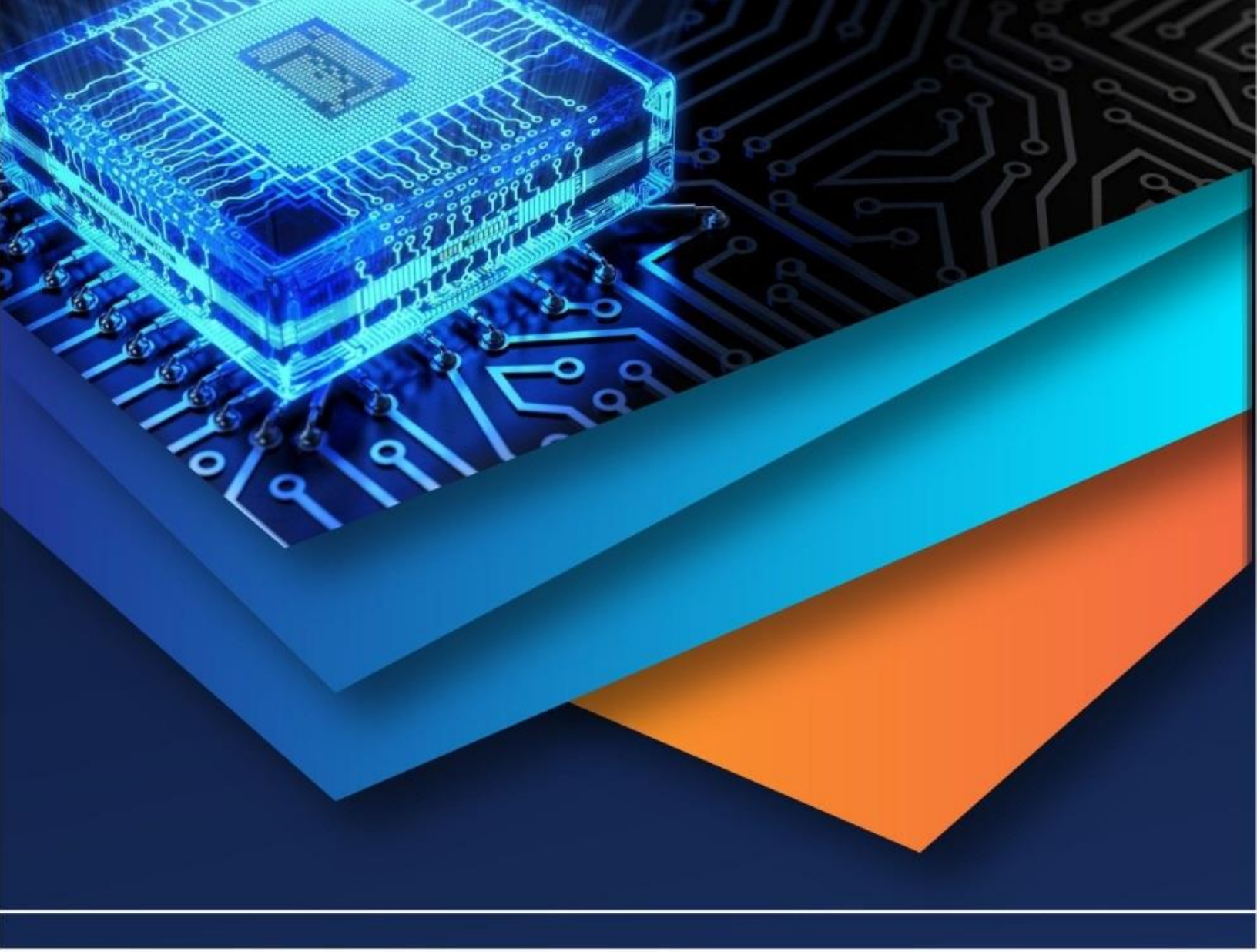
REFERENCES

[1] Begel, A., & Graham, S. L. (2005). Spoken programs. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing.
[2] Pane, J. F., & Myers, B. A. (2001). Usability issues in the design of novice programming systems. IBM Systems Journal, 40(2), 529–549.
[3] Van Brummelen, J., Weng, K., Lin, P., & Yeo, C. (2020). Conversational programming systems and machine learning interface design.
[4] Lucas Rosenblatt. 2017. VocalIDE: An IDE for Programming via Speech Recognition. In Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '17). Association for Computing Machinery, New York, NY, USA, 417–418. https://doi.org/10.1145/3132525.3134824
[5] Mankoff, J., Hayes, G., & Kasnitz, D. (2010). Disability studies as a source of critical inquiry for the field of assistive technology. Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility
[6] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine.
[7] V. Kumar, “A next-generation AI voice assistant for computer: Multilingual, secure, and context-aware interaction,” Asian Conference on Communication and Networks, 2025.
[8] S. Revankar, S. Deshpande, A. Sayeed, and A. Tandale, “Sanvaad: A multimodal accessibility framework for sign language recognition and voice interaction,” arXiv preprint, 2025.
[9] B. H. Juang, “Speech recognition and acoustic signal processing advances,” IEEE Signal Processing Magazine, vol. 39, no. 2, pp. 10-20, 2022.
[10] D. Povey et al., “The Kaldi speech recognition toolkit,” IEEE Signal Processing Society, 2011.
[11] Apple Machine Learning Research, “Improved speech recognition for people who stutter,” Apple Research, 2023.
[12] M. Bond, “Spoken AAC: Assistive communication application for speech-impaired users,” Spoken Inc., 2025.
[13] S. Lee et al., “Intelligent personal assistant implementing voice commands using speech recognition,” IEEE Conference Publication, 2020.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VII July 2026- Available at www.ijraset.com

Mr. D. Shiva Naik is an Assistant Professor at Lenora College of Engineering. He is dedicated to teaching, research, and mentoring students in engineering education. With a strong commitment to academic excellence, he actively guides undergraduate and post-graduate students in their research projects and encourages innovation and technical skill development. His areas of interest include emerging technologies and fostering practical learning through research-oriented activities.

Epili Santosh Kumar Patro is an M.Tech student in the Department of Computer Science and Engineering (CSE) at Lenora College of Engineering, Rampachodavaram, Andhra Pradesh. He is passionate about advancing his knowledge in computer science and emerging technologies. His academic interests include software development, artificial intelligence, and data-driven applications, with a strong focus on research, innovation, and practical problem-solving. He is committed to enhancing his technical skills and contributing to impactful research in the field of computer science.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY