CODE GENERATOR BASED ON VOICE COMMANDS

Mr.CH Vijaya Kumar[1]
Pathakala Rohit[2], Mohammed Fayas[3], Mohammed Ashraf[4], Gandla Bhanu Prakash[5]

[1]Associate professor, Department of Computer Science and Engineering, ACE Engineering College, Hyderabad, Telangana, India. e-mail:vijay.chandarapu@gmail.com

[2,3,4,5] IV B. Tech Students, Department of Computer Science and Engineering, ACE Engineering College, Hyderabad, Telangana, India.
e-mail: rohitrohit51765@gmail.com[2], ashraf17042001@gmail.com[3], adilfayaz01@gmail.com[4],bhgandla32@gmail.com[5]

ABSTRACT

Artificial intelligence is the current focal point of the technology world. Voice assistants with this intelligence are popular in the ai world, including Siri, Google, ChatGPT, and others. Without realising what kind of technology, we are using, but enjoying it, programming plays a vital role in the development of projects like these. However, these demonstrate the significance of programming today. Some innovations take three days or three years to develop. Since the advent of programmable computing systems, programming has been the human race's largest technological challenge.

So, our goal is to find the simplest possible technique to code the programme, which is where project comes in. This project was given the title "Code Generator Based on Voice Commands." In this application, we may communicate with the compiler verbally while we are programming. We may code a programme that uses voice instructions in our project by using a few unique commands. The Python programming language is the foundation of our entire endeavour. Our project's major goal is to provide a user-friendly platform so that programmers may easily implement technologies using voice commands. People who are physically unable to use a computer but are interested in software are most likely to take up this endeavour. They can now code using voice commands thanks to them.

Keywords: Natural Language, Voice Command, Compile, Translate, Instruction

INTRODUCTION

The two digits 0 and 1 developed into the meta world we live in today, giving rise to supercomputers, cutting-edge VFX graphics, and other modern technologies. As we can see, programming is used to create every technology we use today. Because programming is a difficult undertaking, humanity has suffered since its invention. Depending on how many people are working on it to write the code, some technologies take 3 days to build while others take 30 years. To speed up development, our team came up with a solution: a voice-based code generator that makes the work of programmers easier.

Future technology is advancing towards hands-free capabilities, which improves usability. The leading businesses are incorporating assistants like Amazon Echo, Google Home, Siri by Apple, etc. as these concepts gain popularity. According to the voice assistant usage, people are more interested in voice-based technologies. In the field of IT, it also has a great future potential.

Voice-based code generators generate codes based on user voice recognition and voice commands. This project was broken down into four main components for development. Those are

1.user input via voice.

2.handling the input.

3. Producing Code from Input.

4. The Code is compiled.

We used the Google Voice API to capture user voice input. The compilation of our project's code, which comes next, demands a significant storage space for libraries and data.

The Google API is used to interpret voice instructions into sentences, which are subsequently broken down into words. Then, depending on keywords and phrases, our programme understands user commands and generates code. To generate the code, these words and phrases are concatenated using an algorithm. It is less complicated because we used a limited amount of syntax and functions, but the algorithm will be more complicated. External libraries, techniques, and sophisticated sections are generally tolerated.

We used Natural Language Processing to create our project. Where NLP is a subfield of artificial intelligence. Natural Language Processing (NLP)-based voice command code generators enable users to speak their instructions to the system to generate code. To translate voice commands into executable code snippets, it combines speech recognition, NLP, and code generation techniques.

Such a system typically consists of several parts, such as a speech recognition module to translate voice commands into text, a module to extract intent and parameters from natural language, a module to generate code based on user requirements, a module to execute the generated code, and so on.

To accurately record voice commands, the speech recognition system's accuracy is essential. The NLU component must be able to comprehend complex requests for code generation and extract pertinent data. Coding conventions and best practises should be followed by the logic used to generate accurate and effective code. A voice command-based code generator should offer a safe environment for operations and

gracefully handle errors. With a user-friendly interface and clear instructions, usability and user experience are crucial. Users can add or alter code generation templates thanks to flexibility and extensibility.

Performance is essential for handling complex requests quickly, and the system should evolve over time and consider user feedback to continuously improve.

It is crucial to keep in mind that creating a comprehensive and precise code generator based on voice commands is a challenging task that necessitates a thorough understanding of natural language processing, code generation methods, and execution environments. To aid in the language understanding and code generation tasks, take into consideration utilising already-existing NLP frameworks like Hugging Face's Transformers or OpenAI's GPT-3.

SO, we have created this project as a result of how our project affected the outside world. It makes no sense without affecting our project in any way. Making tech programming a quicker and simpler entry point for new interns to programming is one of the key effects that motivated us to work on this project.

PROPOSED SYSTEM

Developers may find it useful and effective to use a voice-activated code generator that produces code snippets or templates. Here is a suggested system architecture for a voice command-based code generator:

Voice Input: The system ought to have a voice recognition feature that can translate verbal instructions into text. For this purpose, a few speech recognition libraries and APIs are available, including Google Cloud Speech-to-Text, CMU Sphinx, and Mozilla Deep Speech.

Command processing: After converting the voice input to text, a command processing module will analyse and draw out the pertinent details from the command. The developer's intent should be understood by this module, which should also determine the desired code snippet or template to be generated. Here, Natural Language Processing (NLP) techniques can be used to improve the precision of command interpretation.

Code Generation: The code generation component creates the corresponding code snippet or template from the processed command. This step might entail integrating with code generation libraries or parsers for programming languages. The code generator ought to be flexible and support a variety of frameworks and programming languages.

Code Output: The user should be shown the generated code snippet or template in an appropriate format. It may be made available as a downloadable file or displayed on a graphical user interface (GUI). The system might also provide customization options, such as for variable names, method signatures, or configuration parameters.

Code Execution: Depending on how complex the generated code is, the system might offer a choice to run the code directly from the integrated development environment (IDE) or might offer guidance on how to add the code to an already-existing project.

Error Handling: It is critical to gracefully address any mistakes or ambiguities that may occur when using voice commands. The system must have mechanisms for detecting errors and giving the user the proper feedback, such as by requesting clarification or recommending different commands.

Integration with IDEs: Integrating the code generator system with popular integrated development environments (IDEs) like Visual Studio Code, IntelliJ IDEA, or Eclipse will improve the developer's workflow. This integration might include add-ons or extensions that enable easy access to the IDE's built-in code generator.

User Feedback and Improvement: It would be advantageous to gather user feedback on the generated code samples and continuously enhance the precision and performance of the system. Users could give feedback on the effectiveness, appropriateness, and quality of the generated code, enabling the system to develop and change over time.

It is important to keep in mind that creating a voice-based code generator requires the development of several different parts, including voice recognition, natural language understanding, and code generation. Both software development and speech processing expertise are needed.

PERFORMANCE EVALUTION

A code generator's performance based on voice commands is evaluated by looking at a variety of factors, including accuracy, speed, usability, and overall efficiency. When assessing the effectiveness of a code generator that responds to voice commands, keep the following points in mind:

Evaluate the code generator's accuracy in translating voice commands into executable code. Assess the system's performance using a variety of inputs and the generated code's accuracy. Check the generated code for any syntax mistakes, logical inconsistencies, or inaccuracies.

Word error rate (WER): This is the most common metric used to evaluate S2C systems. It is calculated as the percentage of words in a test set that are incorrectly recognized.

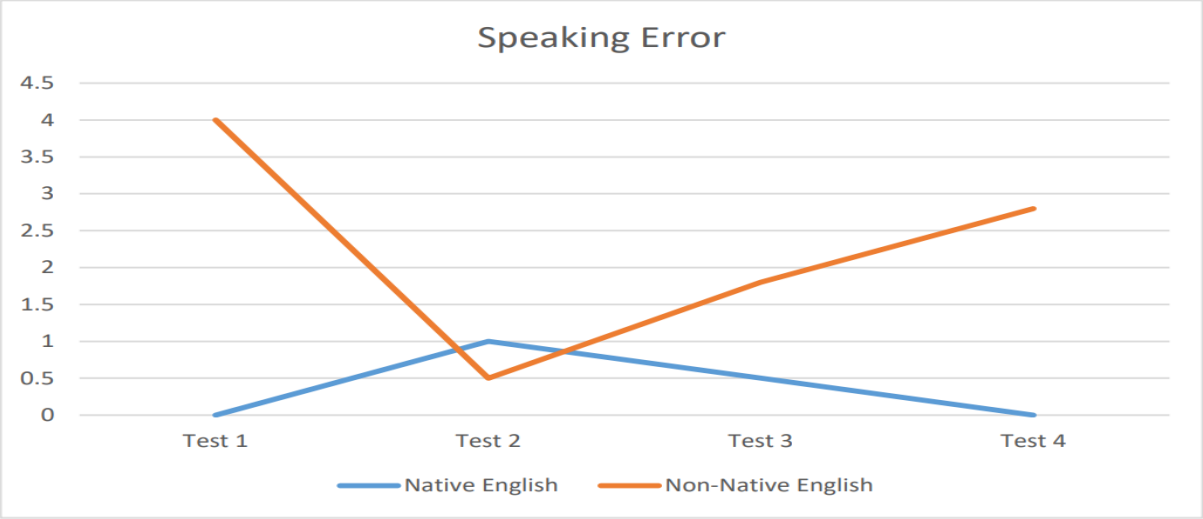

Fig 1: Speaking Error

Speed: Time the code generator needs to translate voice commands into code. Contrast the processing speed with other code generation techniques. Consider elements like code compilation time, voice recognition latency, and overall system responsiveness.

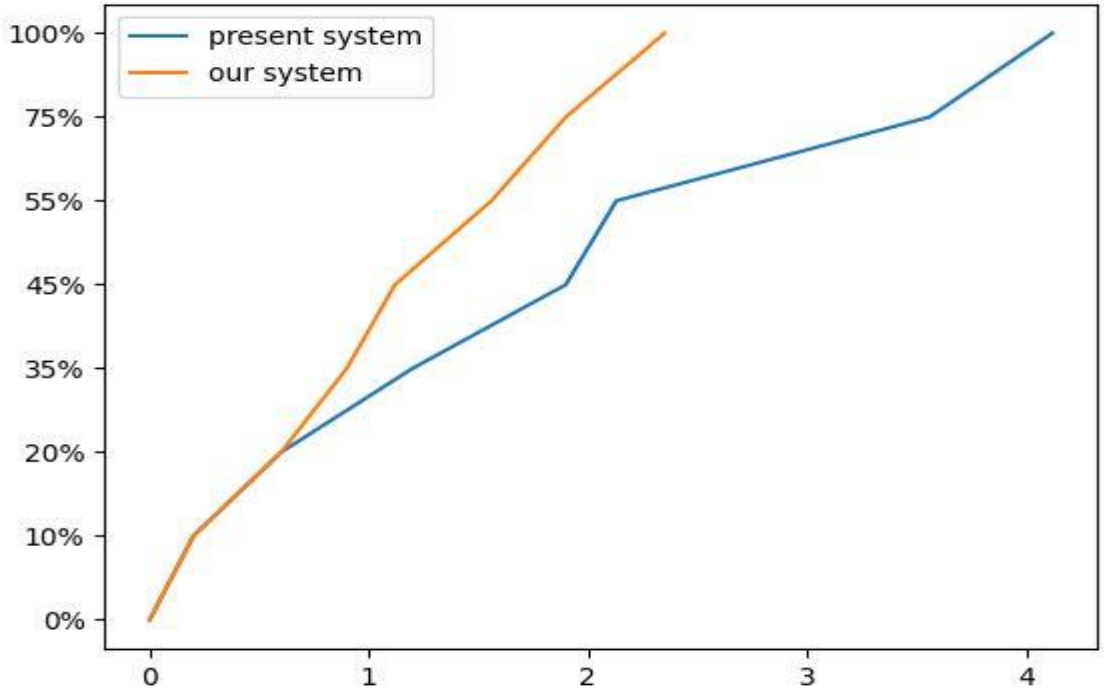

Fig 2: Speed Test in voice recognition

Error Handling: Evaluate the code generator's ability to handle errors or unclear voice commands. When presented with ambiguous voice inputs, check to see if the system prompts for clarification or displays clear error messages. Evaluate how well the code generator handles mistakes or unclear voice commands. When presented with ambiguous voice inputs, check to see if the system prompts for clarification or displays clear error messages.

We can see that most of the mistakes occur in the classroom. Since everyone is aware of how noisy classrooms can be. The voice is essential to our application. For the application to function properly, we need crystal-clear voice input. Because of this, there are more mistakes made in classrooms. On the other hand, we can observe that the home appears to have fewer errors. The cause of this is that homes are generally quieter than classrooms.

CONCLUSION

In conclusion, a code generator based on voice commands has the potential to enhance the programming experience by providing a hands-free and intuitive way to generate code. It can be a valuable tool for developers, especially those who may have physical limitations or prefer a more natural interaction with their programming environment. The advantages of a code generator based on voice commands include Increased productivity: Voice commands can potentially speed up the code generation process by eliminating the need for manual typing. Developers can dictate their code and see it generated automatically, reducing the time and effort required.

Accessibility and inclusivity: Voice-based interfaces can make programming more accessible to individuals with disabilities or physical limitations that make typing difficult. It allows them to participate in software development on an equal footing with others. Intuitive and natural interaction: Voice commands provide a more natural way of interacting with the programming environment. Developers can express their thoughts and ideas verbally, making the process feel more fluid and intuitive. Multitasking capabilities: With a voice-based code generator, developers can generate code while performing other tasks simultaneously. They can dictate code snippets or instructions while focusing on other aspects of their work, such as problem-solving or design.

Contextual understanding: Programming often involves complex syntax, context, and structure. It may be challenging for a voice-based code generator to understand and interpret the nuances of programming languages accurately. It needs to be able to differentiate between code instructions and normal speech to generate correct and valid code.

Limited control and precision: Voice commands may not provide the same level of control and precision as manual typing. Fine-grained adjustments and precise formatting can be more challenging to achieve with voice-based inputs.

ACKNOWLEDGEMENT

We would like to thank our guide, Mr. **CH Vijaya Kumar,** for his continuous support and guidance. Also, we are thankful to our project coordinator, **Mrs. Soppari Kavitha,** and we are extremely grateful to **Dr. M. V. Vijaya Sardhi**, Head of the Department of Computer Science and Engineering at Ace Engineering College, for his support and invaluable time.

References:

1. Goldthwaite, John, Mark, and Stephen Arnold. Vocal programming, or voice programming. In the ACM 2000 Conference on Assistive Technologies Proceedings, November 2000.

2. Lindsey Snell. Development of a Toolkit for Writing Voice-Controlled Applications and Research into Voice Programming. Engineer's Report. June 2000, Imperial College of Science, Technology, and Medicine in London

3. Peter Bednr, "Vocabulary matching for information extraction language, EEE 15th International Symposium on Applied Machine Intelligence and Informatics," Herlany, Slovakia, January 26–28, 2017.

4. "Keyboard less visual programming using voice, handwriting, and gesture" by Jennifer L. Leopold and Allen L. Ambler.

Unified parsing and information extraction language, 5.P. Bednr. January 21–23, 2016, pp. 131–135 in SAMI 2016 – IEEE 14th International Symposium on Applied Machine Intelligence and Informatics.

6. Mr. Ing. Marilena Anghelu and Mr. Ing. Alexandru Trifan.Rodica Constantinescu, L. Dr. "Compiling a Natural Language Processing Model. into Byte Code from Natural Language.

7. Jeff Grey and Amber Wagner. An Empirical Assessment of a Vocal User Interface for Voice-Activated Programming.

8. G. S. (2016, May 24). According to Google, voice searches account for 20% of mobile queries. February 15, 2018, retrieved.