INTERNATIONAL JOURNAL OF
RECENT TRENDS IN MULTIDISCIPLINARY RESEARCH
https://www.doi.org/10.59256/ijrtmr.20240403007
May-June 2024, Vol 4(03), 29-34.
ISSN No: 2583-0368

A Smart Interactive Programming Assistant for Error-Free Coding

Jerold Jacob T[1], Anjana S[2], R J Arunasree[3], Vasuki P[4]
[1,2,3,4] *Dept of Information Technology, Sri Sivasubramaniya Nadar College of Engineering, Chennai, Tamil Nadu, India.*

OPEN ACCESS
Article Citation:
Jerold Jacob T[1], Anjana S[2], R J Arunasree[3], Vasuki P[4], 'A Smart Interactive Programming Assistant for Error-Free Coding", International Journal of Recent Trends in Multidisciplinary Research, May-June 2024, Vol 4(03), 29-34.

Abstract: This paper is aimed to make the programming easier and user friendly to the beginners. The programming tools use compilers which convert high level language to low level language. For efficiency the code should be error free and optimized code. Beginners find it very difficult to resolve the errors and to optimize their code. This project proposes an intelligent interactive programming tool which guides the programmers by correcting the basic errors they commit. The system provides a speech interface where the speech is converted into text with proper syntactic arrangements and resolves ambiguities. In addition to this, common syntactic errors made by the users can be identified interactively and are highlighted, thus helping the users to do programming in a user-friendly way. It uses Google API for speech recognition and its converted to the syntactic form of C++ programming language. Errors are highlighted by first splitting errors using java regular expressions and then highlighting to identify the common errors that occur during compilation. Therefore, this project will be very much helpful not only for novice programmers to correct the errors in the early stage with interactive comments.

Key Words: Speech Recognition, programming languages, compiler construction, verbal programming

1. Introduction

Programming is a fundamental aspect of the curriculum for engineering students, particularly those specializing in computer science. As they delve into various programming languages, such as C++, they encounter challenges associated with comprehending object-oriented concepts, especially for those new to programming. Additionally, navigating sophisticated programming environments and understanding the role of compilers, which translate high-level programming languages into machine code through processes like preprocessing, lexical analysis, parsing, semantic analysis, and code optimization, can prove daunting for novice programmers.

Beyond the realms of programming, speech stands as the primary mode of human communication, finding applications across diverse fields. Its increasing prominence in research underscores its potential utility in programming environments, promising a user-friendly interface. Automatic Speech Recognition (ASR) technology, enabling computers to capture spoken words via microphones, plays a pivotal role in this regard. However, the efficacy of speech recognition systems hinges on factors like vocabulary, ambient noise, and concurrent users operating from different workstations.

A burgeoning area of research involves programming by voice—a concept aimed at addressing the needs of individuals with exceptional programming skills but hindered by physical constraints such as Repetitive Strain Injury (RSI). A substantial segment of the global population—approximately 30%—faces various disabilities, including blindness or impaired hand function, rendering traditional text-oriented programming tools and editors inaccessible. In such cases, speech recognition systems offer indispensable support, allowing users to input code through voice commands.

The prevalence of RSI, characterized by discomfort or difficulty in typing due to repetitive tasks, underscores the significance of accommodating diverse user needs in programming environments. Helliwell and Taylor (Reference [1]) elucidate the causes of RSI, attributing it to repetitive motions, excessive force, and prolonged unnatural postures associated with certain work activities. By acknowledging and addressing such challenges, this research endeavors to explore the integration of speech recognition technology into programming interfaces, fostering inclusivity and facilitating seamless engagement with computational tasks for all users.

2. Related Work:

The existing system has numerous IDEs and code editors with speech enabled features, but beginners often face difficulties in compiling and running programs due to unfamiliarity with C++ syntax. There are few voice-based coding platforms for languages like Java, Python, and C. Speech Clipse, developed by Jeff Gray, is a speech-enabled[1] Eclipse programming environment, but not the text editor [15]. Voice Code, a success using finite-state command grammars, supports Python and C++. Natural Java accepts Natural Language descriptions of programs, but only supports code authoring and speech editing. Coders specialized in C++ have limited options compared to those coding in other languages.

Coding Skills.

When it comes to errors in C++, they can be categorized into three types: compile-time errors, runtime errors, and semantic errors. Various technologies are employed to address, detect, and rectify these errors, including methodologies outlined in research such as "Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks" [2]. This research proposes a method for automatically generating corrective feedback for syntax errors encountered in introductory programming problems, leveraging recurrent neural networks. The findings indicate that this approach is capable of fully rectifying syntax errors in 31.69% of submissions and partially correcting an additional 6.39% of submissions.

.Programmers often encounter challenges in identifying both syntactical and logical errors during program execution. To address this issue, researchers have explored techniques involving machine learning and data mining for error detection. A novel model is proposed in this regard, leveraging machine learning and data mining methodologies to detect and correct syntactical and logical errors, or provide suggestions for their resolution. The proposed approach utilizes hashtags to uniquely identify each correct program, facilitating comparison with potentially flawed programs to pinpoint errors [3]. Additionally, runtime errors are detected through source code implementation, wherein a ROSE source-to-source compiler is introduced for instrumentation and comparison with conventional binary instrumentation tools [4].

People with hand injury may feel hard to type in normal code editors that are available which are text-oriented. This realization is given by the Spoken Java Programming by Voice: A Domain-specific Application of Speech Recognition [5]. Here, a dialect of Java is proposed that is naturally verbalized by human programmers which is identical to Java yet the difference is the form of input which is speech here. This presents the result which is indistinguishable from a traditionally coded Java program.

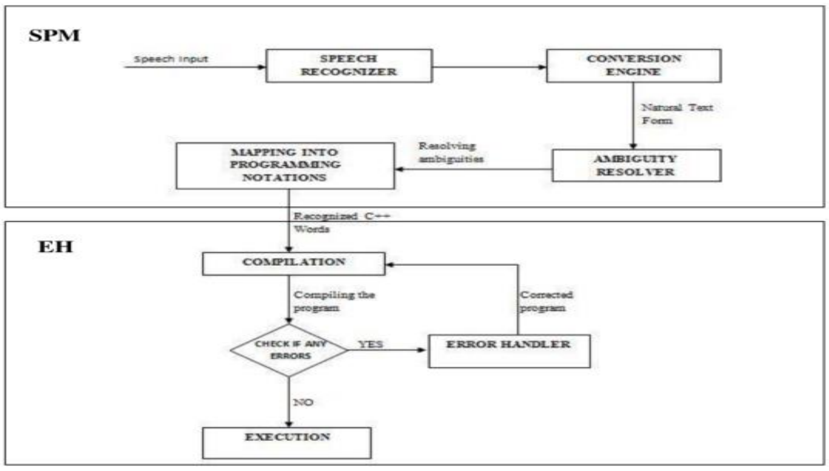
We specifically concentrate on an individual programming language C++. It is said that coding through voice is faster in some cases and proves to be efficient if it is trained adequately. This realization is provided by a software Voice Code Java Programming Using Voice Input [6]. This focuses particularly on java where voice code, a software, is used with editors like emacs along with Dragon naturally speaking system.

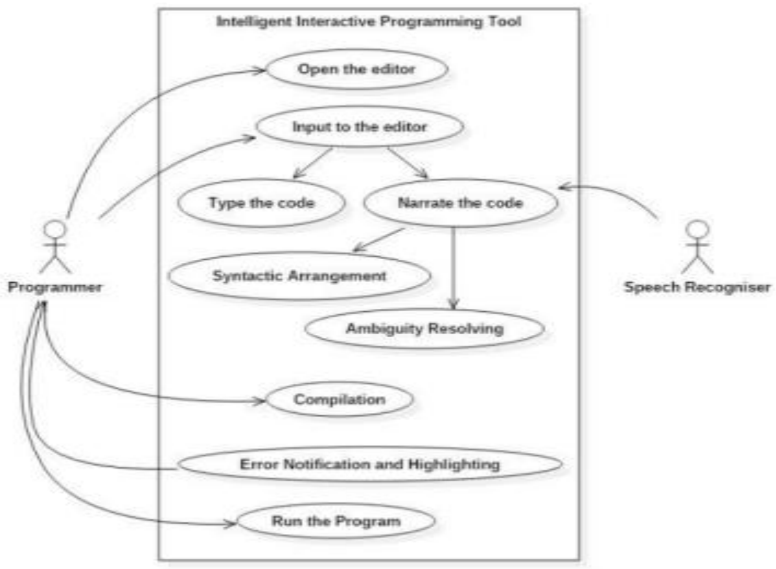
3. Methodology

The system has been meticulously crafted to encompass two pivotal modules: the Speech to Program Module (SPM) and the Error Handler (EH). Within the SPM lies a robust Speech Recognition engine, seamlessly integrated into the project through the utilization of prebuilt Google API. This engine adeptly captures speech input and swiftly translates it into textual form. The recognized text undergoes meticulous mapping to generate program code in a specific programming language. For instance, the inclusion of recognized text is intelligently transformed into the corresponding directive, such as converting "has(h) include" to "#include". Users are granted the flexibility to fine-tune the generated program using a conventional keyboard, should the need arise.

The system accommodates dual input methodologies: users may opt to articulate their code through voice commands or employ traditional keyboard typing. In either mode, the input is swiftly translated into executable program code. Subsequently, the compiled program is transmitted to the Error Handler (EH) module for further processing. In the event of detected errors, they are promptly highlighted, and informative tooltips are provided to aid users in rectifying the issues. Certain minor errors, such as missing semicolons or inadequate spacing between keywords and variables, are automatically rectified with the user's consent, streamlining the programming experience.

The modules are explained in the block diagram as follows:





The proposed system is designed with a simple user-friendly code editor. developed using NETBEANS IDE. The editor gets speech input and converts into program as well highlighting the compilation errors to ensure easy error correction. Our project utilizes Google Speech Application Programming Interface (API) to convert voice to text. Ambiguities in programming notations are also resolved using the same.

The system uses Google Speech API for speech recognition. Google Cloud Speech API enables developers to convert audio to text by applying powerful neural network models in an easy to use API. The API recognizes over 110 languages and variants, to support the global user base. It can transcribe the text of users dictating to an application's microphone, enable command-and-control through voice, or transcribe audio files, among many other use cases [10]. It recognizes audio uploaded in the request, and integrates with the audio storage on Google Cloud Storage, by using the same technology Google uses to power its own products.

There are many models, methods and algorithms which are used in the speech recognition process. Some of them are as follows:

- Hidden Markov Models
- Dynamic time-warping based speech recognition
- Neural Networks
- Deep feed-forward and recurrent neural networks
- End-to-End automatic speech recognition.

3.1.1. Speech Using Google Api

Google speech API uses powerful neural networks to convert speech to text. Some features of the API are as follows:

The Automatic Speech Recognition (ASR) system, powered by advanced deep learning neural networks, facilitates voice search and speech transcription across over 110 languages and variants. Real-time recognition results are provided as users speak, with customization options available to tailor recognition to specific contexts. This includes incorporating sets of words and phrases relevant to the application's domain, thereby enhancing vocabulary with custom words, names, and industry-specific terminology. The ASR solution accommodates diverse audio input sources, captured through the application's microphone or obtained from pre-recorded audio files, supporting various audio encodings such as FLAC, AMR, PCMU, and Linear-16. Furthermore, the system excels at handling noisy audio without requiring additional noise cancellation measures. Integration with Google Cloud Storage enables seamless management and retrieval of audio files for transcription purposes.

3.1.2 Speech Recognition Using Neural Networks

Neural networks emerged as a compelling acoustic modeling approach in Automatic Speech Recognition (ASR) during the late 1980s. Since then, they have been extensively employed in various facets of speech recognition, including phoneme classification, isolated word recognition, audiovisual speech recognition, audio-visual speaker recognition, and speaker adaptation.

In contrast to Hidden Markov Models (HMMs), neural networks do not rely on assumptions about feature statistical properties and possess several appealing qualities as recognition models for speech. When utilized to estimate the probabilities of speech feature segments, neural networks enable discriminative training in a natural and efficient manner. Furthermore, they impose few assumptions on the statistics of input features. Recently, Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs) and Time Delay Neural Networks (TDNNs) have gained prominence, demonstrating their ability to identify latent temporal dependencies and effectively utilize them in the recognition process.

3.1.3 Deep Feed Forward and Recurrent Neural Networks

A deep feed-forward neural network (DNN) is characterized by its architecture comprising multiple hidden layers of units situated between the input and output layers. Similar to shallow neural networks, DNNs excel in modeling complex nonlinear relationships within data. The architectural design of DNNs fosters the creation of compositional models, where additional layers facilitate the integration of features from lower layers. This attribute endows DNNs with substantial learning capacity, enabling them to effectively capture intricate patterns present in speech data.

DNNs represent a departure from conventional artificial neural networks (ANNs) and are increasingly recognized as efficient tools for addressing contemporary real-world challenges. Particularly in the domain of speech recognition, DNNs have emerged as a mainstream solution, rapidly gaining prominence and supplanting traditional methods. Leveraging deep learning techniques, which employ a generative, layer-by-layer pre-training approach for initializing weights, DNNs have demonstrated unparalleled efficacy in acoustic modeling for speech recognition tasks.

The method involves ingesting lots of data to train systems called neural networks, and then feeding new data to those systems in an attempt to make predictions.

A success of DNNs in large vocabulary speech recognition occurred in 2010 by industrial researchers, in collaboration with academic researchers, where large output layers of the DNN based on context dependent HMM states constructed by decision trees were adopted.

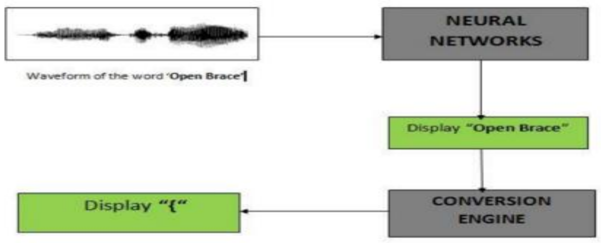


Figure shows the process of speech recognition using neural networks.

The waveform of a particular word is sent as input to the neural network which displays words in natural language. This displayed word is given to the conversion engine which converts the word to its appropriate programming notation.

3.1.4 Conversion To Programming Notations

Once the input speech is converted to text it is displayed as natural plain text, For example, the open brace after a class declaration may be verbalized as "open brace," but for our application we need to convert it to programming notations confirming to the syntax of the programming language C++, by resolving ambiguities in the conversion from speech to text. For eg: there may be ambiguities like the way one pronounces a keyword. Spoken programs are full of ambiguities caused by unverbalized punctuation. For example, when users want to extract an element of an array at a certain index, they write "a[i]". Most of the time, however, when users speak the same construct, the right bracket is not verbalized (e.g. "a sub i"), leading to an ambiguity.

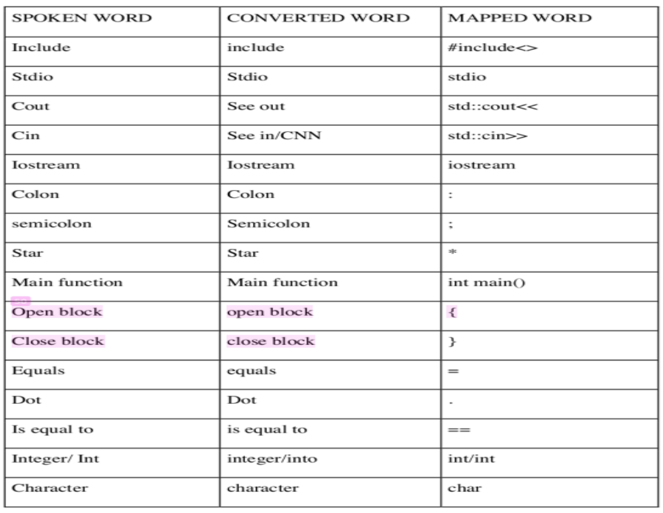
Another important one arises from homophones, words that sound alike but are spelled differently. Here is an example: a programmer wishes to enter a loop construct to count from one to ten. In Java, the result should come out like this:
for(int i=0;i<10;i++){}

Typically, a programmer would verbalize the preceding code fragment without any parentheses, braces or semicolons, and thus say just the following words: for int i equals zero if less than ten i plus plus

These ambiguities need to be resolved by mapping patterns (i.e. that is converting them to appropriate symbols). This is done through a java program.

Table 3.1 shows some of the converted words from normal text to programming notations. After converting to program notations, it is corrected according to the syntax of C++ as shown in Table 3.1.

Figure 3.1 shows the process of speech recognition using neural networks. The waveform of a particular word is sent as input to the neural network which displays words in natural language. This displayed word is given to the conversion engine which converts the word to its appropriate programming notation.



SPOKEN WORD	CONVERTED WORD	MAPPED WORD
Include	include	#include<>
Stdio	Stdio	stdio
Cout	See out	std::cout<<
Cin	See in/CNN	std::cin>>
Iostream	Iostream	iostream
Colon	Colon	:
semicolon	Semicolon	;
Star	Star	*
Main function	Main function	int main()
Open block	open block	{
Close block	close block	}
Equals	equals	=
Dot	Dot	.
Is equal to	is equal to	==
Integer/ Int	integer/into	int/int
Character	character	char

Empirical Regression Models

Regression modeling refers to a class of methods used for building black box models of systems. Say we are given a system defined by a set of input parameters and an output variable (also known as the response). Assume that the system is described by some mathematical function, which represents the relationship between the inputs and the response. In regression modeling, we want to find an approximation of this function solely by observing the response of the system for different inputs.

Let us state this notion Formally. We are given a system with input parameters $\{x_i \mid 1 \leq i \leq n\}$and a response variable y. We are also given a set of observations {(X1, y1), (X2, y2), . . . , (Xm, ym)}, where each Xi is an n-dimensional vector and represents an assignment of values to the input parameters, and yi is the response of the system for the input Xi .

3.2 Error Handler

Also assume that the system is characterized by the following equation:

$$y=f(x1,x2,...,xn)$$

In regression modeling, we use the observations to learn a function $\hat{f}$ that satisfactorily approximates f .
Formally,
$y= f(X)= \hat{f}(X)+\varepsilon$

where ε represents the approximation error. One can think of regression modeling as the process of searching for a function that best explains the observations. However, since there are infinitely many functions, the search for the closest approximation may never terminate. Regression modeling techniques restrict the search space by making some assumptions about the structure and form of the approximation function $\hat{f}$. We now describe two techniques, linear regression models [25] and multivariate adaptive regression splines (MARS) [26], that are commonly used for building regression model**s.**

Adaptive Compiler

The compiler uses optimization techniques. The code gives suggestions to give a suggestion while the programmer codes. The system provides suggestions when the un optimised code is given and corrects when erroneous input is given. The compiler follows the adaptive architecture proposed by —- as shown in the figure. The suggestions and modifications are taken into consideration based on the user consent and the changed code is given as an input of the compiler.

Once the program is saved, it moves into the compilation phase. In order to highlight any particular error, it must be stripped by its type, line number, column number. Regular expressions are used to strip the errors since it provides an easy way to extract the required information from the error messages. Regular expressions are used with a pattern similar to the output error messages which when matched again provides the extracted output. This output contains the line number and the column number where the error has occurred and also the type of error message. Using this line number, the particular error can be highlighted with a red colored line in the text editor. This helps the users to locate and correct the errors in a much easier way.

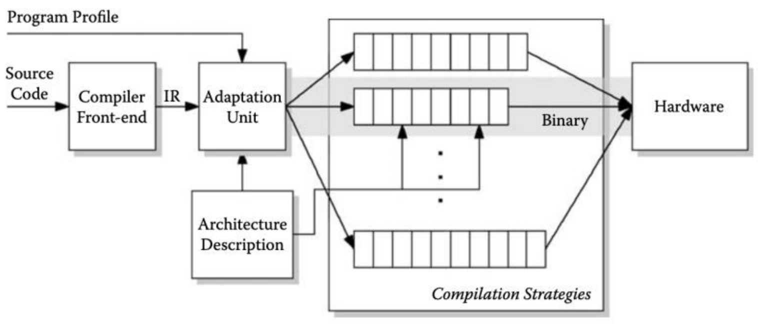


FIGURE 8.5 The architecture of an adaptive optimizing compiler.

Implementation

This work is implemented on Netbeans IDE, where a simple code editor is designed which compiles and runs C, C++ code using GNU Compiler Collection(GCC) Compiler. Here the input is given in the form of speech using a microphone which is converted into text using Google API, next the ambiguities are resolved, converted to programming notations confirming the basic syntax of C++. After compilation, in case there are errors then it is highlighted distinctly in a different color so that the programmer can identify easily according to the line number displayed in the text area with the description of errors given in the error box. Then the program is compiled again and executed.

4. Conclusion

In conclusion, the integration of speech recognition technology into programming environments offers a promising avenue for enhancing accessibility and usability, particularly for individuals facing physical challenges such as repetitive strain injury or visual impairments. Programming by voice enables users to interact with computers and write code without relying solely on traditional keyboard input, thereby mitigating the limitations imposed by text-oriented interfaces. Furthermore, automatic speech recognition systems continue to advance, offering improved accuracy and performance across diverse environments and user scenarios. By harnessing the power of speech technology, we can foster a more inclusive and user-friendly programming experience, empowering individuals with disabilities to participate fully in the field of computer science and engineering.

In conclusion, the novelty of integrating speech recognition technology into programming environments lies in its ability to address accessibility and usability challenges, foster inclusivity, and leverage advancements in speech recognition systems to create a more intuitive and user-friendly programming experience.

Reference

1. *Raza, Hafiz Yasar, et al. "Speech Recognition Based on Hidden Markov Models: A Review." EURASIP Journal on Advances in Signal Processing, vol. 2014, no. 1, 2014.*
2. *Mandal, Ranjit Kumar, and Saroj Kumar Panigrahy. "Design and Implementation of Voice-Based Software Application Using Visual Basic." International Journal of Computer Science and Mobile Computing, vol. 2, no. 10, 2013, pp. 1-8.*
3. *Kim, Byeongchang, et al. "Voice Code: Voice-Enabled Coding Environment." Proceedings of the 14th International Conference on Ubiquitous Computing, 2012, pp. 179-188.*
4. *Wang, Xu, et al. "JVoiceXML: A Design of Java-Based VoiceXML Interpreter." Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing, 2006, pp. 1171-1181.*
5. *Nayak, Prasant Kumar, and Sandeep Kumar Satapathy. "Voice Controlled Device Using MATLAB." International Journal of Computer Applications, vol. 54, no. 16, 2012, pp. 38-42.*
6. *IBM Corporation. "IBM Watson Speech to Text." IBM Developer, 2022, https://developer.ibm.com/technologies/speech-to-text/.*
7. *Google LLC. "Speech-to-Text." Google Cloud, 2022, https://cloud.google.com/speech-to-text.*
8. *Microsoft Corporation. "Azure Speech Services." Microsoft Azure, 2022, https://azure.microsoft.com/en-us/services/cognitive-services/speech-services/.*
9. *Mozilla Corporation. "DeepSpeech." Mozilla DeepSpeech, 2022, https://deepspeech.mozilla.org/.*
10. *Vasquez, Francisco M., et al. "Fairness and Transparency in Artificial Intelligence." arXiv preprint arXiv:2010.04544, 2020.*
11. *Howard, Jeremy, and Jeffrey Sorensen. "The Rise of AI: How Artificial Intelligence is Shaping Computing." IEEE Computer Society, vol. 53, no. 1, 2020, pp. 20-26.*
12. *Wachter, Sandra, Brent Mittelstadt, and Chris Russell. "Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR." Harvard Journal of Law & Technology, vol. 31, no. 2, 2018, pp. 841-887.*
13. *Goyal, Sandeep, et al. "An Overview of Compiler Optimization Techniques for Deep Learning." IEEE International Conference on Big Data, 2019, pp. 5633-5642.*
14. *Lee, Jongsoo, et al. "Optimization of Compiler for Big Data Computing." Proceedings of the International Conference on Computational Science and Its Applications, 2015, pp. 471-486.*
15. *Dr.Manoj Kumar, Hitesh Yadav ,Jayant Yadav, Mayank Jha. "IntelliCode: A speech-based programming environment". Journal of Xi'an Shiyou University, Natural Science Edition VOLUME 19.*