A Compiler-based Approach for Natural Language to Code Conversion

Sashank Sridhar, Sowmya Sanagavarapu
Department of Computer Science and Engineering
College of Engineering Guindy, Anna University
Chennai, India
sashank.ssridhar@gmail.com, sowmya.ssanagavarapu@gmail.com

***Abstract*—There is a gap observed between the natural language (NL) of speech and writing a program to generate code. Programmers should know the syntax of the programming language in order to code. The aim of the proposed model is to do away with the syntactic structure of a programming language and the user can specify the instructions in human interactive form, using either text or speech. The designed solution is an application based on speech recognition and user interaction to make coding faster and efficient. Lexical, syntax and semantic analysis is performed on the user's instructions and then the code is generated. C is used as the programming language in the proposed model. The code editor is a web page and the user instructions are sent to a Flask server for processing. Using Python libraries NLTK and ply libraries, conversion of human language data to programmable C codes is done and the code is returned to the client. Lex is used for tokenization and the LALR parser of Yacc processes the syntax specifications to generate an output procedure. The results are recorded and analyzed for time taken to convert the NL commands to code and the efficiency of the implementation is measured with accuracy, precision and recall.**

Keywords—automatic code generation, speech to code, natural language recognition, natural language processing

I. INTRODUCTION

To bridge the gap between natural language and programming [1], [2] there is a need for the development of an application that could perform the translation seamlessly in real-time. This is proposed here to be done both with speech and text recognition. Natural Language processing(NLP) has been used since language plays a key role in human intelligence and its successful inference for applying in a wide variety of applications [3]. One of the applications of NLP is a natural language based code generator [4]. Natural language involves both speech and text.

Speech to code conversion [5] would tremendously speed up the programming concept and has applications in building Virtual Assistants to the programmers [6]. Speech to code conversion is done using feature selection on the input signal from the user and classifying them [7]. The classified keywords are then converted to text for further processing. Text to programming language conversions has helped in simplifying and relaxing the syntactic standards required by the programming languages and replacing them with human interact-able forms [8].

Compilers help convert user programs into machine level code [9]. Compilers perform lexical, syntax and semantic analysis to generate intermediate code. The intermediate code is then optimized to give machine level instructions.

The proposed system also adopts the same structure of a compiler to convert natural language into user programs. Lexical, syntactic and semantic analysis [10] is done for the input text.

The lexical analysis takes in the input text and processes it to identify the individual words as tokens, depending on the specified regular expressions or grammars. Lexical Analysis in the proposed system is done using Lex. Lex [11] is used to write programs whose control flow is directed by regular expression instances in the input stream. It has a source table of regular expressions and corresponding program fragments. The recognition of the input as a form of regular expression is performed using Deterministic Finite Automata(DF) performed by Lex. The tokens or "lexemes" from the lexical analyzer are then sent to the parser, for syntactic analysis. Syntactic analysis uses parse trees for determining the right form of the sentence by using Context-Free Grammars and this process is called parsing. Syntax trees are used to filter out the redundant information due to implicit definitions. Syntax analysis in the proposed method is done using the Natural Language Toolkit(NLTK) of python [12].

Semantic analysis helps check aspects not related to syntactic forms and not determined during parsing. These include number of variables, type checking of variables and so on. When an input given to the lexical analyzer matches the regular expressions specified, the corresponding code segment is generated. This generation of correct code fragments is done using Yacc. Yacc [13] is a Look-Ahead Left-to-Right (LALR) parser with right-most derivation producer with 1 look-ahead. It takes as input a specification of a syntax and gives as an output the procedure for recognizing that language.

C programming language [14] is a procedural programming language that is used for creating software programs to supercomputers and embedded systems. It is a general purpose and imperative programming language [15] which has unprecedented use in our building compiler-based systems. This paper aims to propose a model to convert natural language to C code. The proposed model would help to create easier large-scale applications using C codes. The compiler-based approach in this paper is independent of the operating system where it is deployed on and uses Python with a Flask architecture. The evaluation metrics are done to compare how many instructions were mistranslated or not translated in the NL commands from the user [16]. Time taken for conversion of natural language to code is also analysed.

The remainder of the paper is organized as follows. Section II outlines the related work. In Section III, we present our proposed system and the methodology followed by detailed implementation in Section IV. Section V presents the

978-1-7281-8247-6/20/$31.00 ©2020 IEEE

results and comprehensive experimental evaluation. Finally, we conclude in Section VI with the proposed future work.

II. Related Work

There have been models proposed for employing different natural language processing techniques for code generation which have been summarized below.

The code generator in Imam et al [16] uses natural language processing techniques to analyze pseudocode and a semantic rule-based mapping machine to perform the composition process. To specify a verb's class when it is present in the pseudocode, a software tool SRL is used in a .Net as an integrated environment. The input given by the user is not in a human interactive form we used here, rather it accepts pseudocode for conversion with multiple pseudo codes possible for a single algorithm. Rahit et al [17] implemented an LSTM model for the conversion from natural language to code by using vocabulary generation, where texts were converted into computational entities, each for source texts and the output codes. The model is limited by the data in the parallel corpus, leading to producing even incoherent codes as output. For the mapping, we used AI with Lex and Yacc compiling. Kulkarni et al [18] used Java libraries to map linguistic constructs to programming structures for creating codes from a given problem statement. The database should be vastly built with available functions as it is limited by the data fed for processing and mapping in SPARQL query implementation. The time taken for the query to fetch the results and the query structure uses standard input conditional, but in the system here we employed Lex that maps dynamic input command to static structure.

Price et al [19] has used Abstract Syntax Tree (AST) to convert natural language to Java codes in the system, NaturalJava. They have used components Sundance to classify the key frames in the NL commands and TreeFace to maintain the AST updated. The model is severely limited by the key frames in the Sundance component and the depth-first development strategy with no compiler or debugger feedback. Trifan et al [20] built a model that uses the AST (Abstract Syntax Tree) with neural networks and genetic algorithms for better performance. The AST is logically validated and the matching Context Free Grammar (CFG) is chosen using the rules and metadata presented. The specific rule set is limited to the grammar validations given to the system and the CFG chosen for that operation. We used LALR parser here instead of tree-based mapping with compiler feedback for removing syntax errors in NL commands.

Schlegal et al [21] built Vajra, an End user Programming (EuP) which employs iterative semantic parsing in End user Programming (EuP) models. The semantic parsing is limited by the vocabulary problem [22] and the further mapping of commands. Observations were recorded on how fast both programmers and non-programmers learnt the system by measuring their time taken to complete the tasks which is highly dependent on the chosen task and participants. Here, we reduced dependency on the user and used a more generic structure for the NL commands given to the system.

In this section, the existing frameworks and their potential limitations along with the advantages of the proposed system were discussed.

III. System Design

The proposed system involves users first interacting with the system either by typing out or dictating instructions. If the user is speaking, the system first listens to what the user tells and then converts the listened lines into textual instructions. After receiving the textual instructions, lexical analysis is used to extract the necessary keywords. Syntax analysis helps to map the text instruction to a code construct. Semantic analysis helps to maintain a count of variables as well as perform type checking when required. By using lexical, syntax and semantic analysis, code is generated. Fig. 1 shows the flow when a single instruction is to be converted to corresponding code.

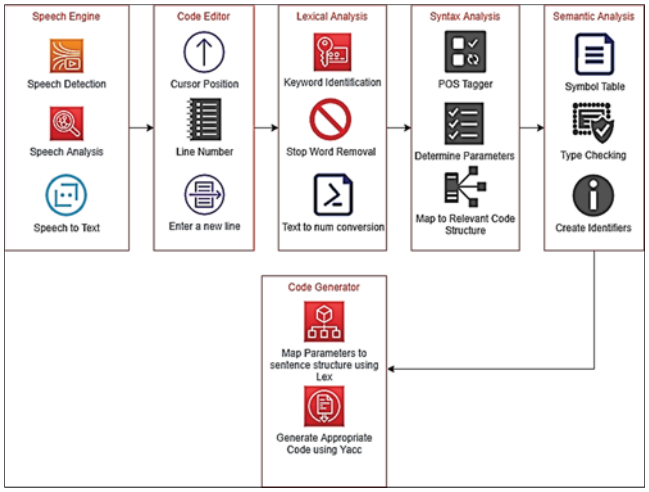

Fig.1 System Architecture with Modules

A. Speech Engine

1) Speech Detection:

Voice Activation Detection(VAD) [23] is the technique by which a system identifies the presence or absence of voice input. It involves identifying whether the user has spoken or not and then sending the frames to the speech recognition module. Voice has to be detected with reduced levels of noise to increase the accuracy of speech recognition. The starting and ending of speech is detected. Fig. 2 shows the working of a VAD system. Segments of speech are determined by extracting features such as energy and duration [7]. Segments which don't match the constraints on the extracted features are rejected as non-speech segments.

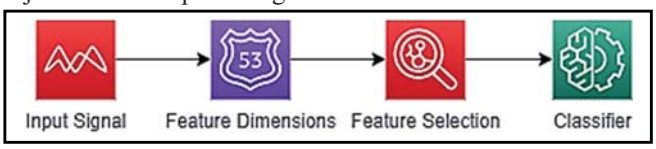

Fig.2 Speech Detection Module

2) Speech Analysis:

When the user speaks, the voice features are extracted [24] and passed to the Speech to Text module. Frequency bands of speech are separated. The volume and speed of the speech signal is normalized. Then the speech signal is broken into segments to match phenomes of the language.

3) Speech to Text:

Fig. 3 shows how an Automatic Speech Recognition Engine works [25]. An ASR does pattern recognition by labelling each speech utterance. Audio signals detected are filtered to remove noise.

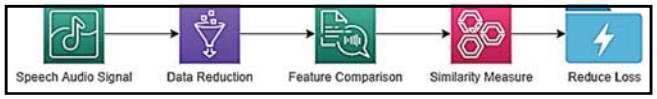

Fig.3 Speech to Text Module

The features of sound waves such as frequency, amplitude and pitch are analyzed and similarity is used to determine which wave the sound signal represents. The prediction of words is made to be most accurate by ensuring the loss is minimum.

B. Code Editor

The code editor is the interface where the user can view the commands given and the related code generated. Users can navigate within the code generated to make corrections or to point where the new code should be added [26].

C. Lexical Analysis

1) Keyword Identification

Keywords in the proposed system are those words which help the system to identify which coding construct to use. Examples of keywords are for, while, if, else and so on. Once the system is able to detect the keywords, it can work towards identifying the parameters required to generate the code snippet.

Fig. 4 shows how keyword identification works [27]. A list of candidate keywords is accumulated, from which the word score of each word is calculated. The word score is the ratio of degree and frequency where degree is the number of co-occurrences of the given word with other words in candidate keywords and frequency is the total number of instances of the given word in the list. The threshold chosen is the mean of all candidate keywords. The candidate keywords are filtered and those keywords whose scores are above the threshold are selected.

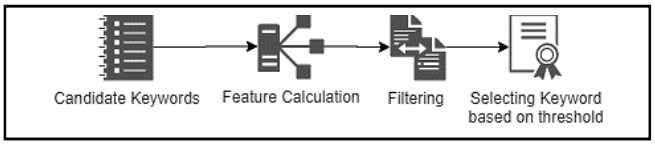

Fig.4 Keyword Identification Module

2) Stop Word Removal

Stop words don't have any significance in understanding the semantic meaning of the sentence and so they can be removed to speed up processing [28]. The given user command is split into words and those words which are present in the list of stop words are removed.

3) Text to Number Conversion

Whenever the user wants a number to be printed, the text based representation retrieved from the Speech to Text module has to be converted into number based representation.

D. Syntax Analysis

1) Parts of Speech Tagging:

Fig. 5 shows the process of POS tagging [29]. The input command given by the user is tokenized by the lexical analyser. On each token, a POS tag is assigned. Disambiguation helps determine what the tag is when the word can be assigned more than one tag.

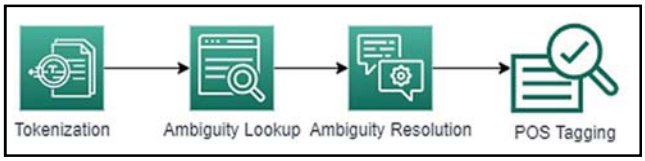

Fig.5 POS Tagging in Syntax Analysis

2) Determine Parameters:

Once the tags are assigned for different words, the parameters for the specific code construct is determined. The nouns, decimal numbers, verbs to determine a particular action such as check or iterate, can all be examples of parameters passed to the code construct.

3) Map to Relevant Code Structure:

The keywords determined in the lexical analyzer and the parameters determined in the syntax analyzer are used to map the user input to a particular code construct. This helps in abstraction of the processing from the code generation.

E. Semantic Analysis

1) Symbol Table:

A symbol table [30] consists of the names of all variables referenced in the coding sequence. This keeps a track of number, type and value of various variables created. It helps the system to check if the variable is already utilized before or if a new variable is to be created.

2) Type Checking:

When a variable is utilized in a code construct, the type of the variable should be known to ensure that the code construct is semantically valid. Using the symbol table, type checking is done for variables [31]. Type checking enables reusability of variables.

3) Creating Identifiers:

Whenever a user creates an identifier, its name, type and value are stored in the symbol table. This helps in easy access and retrieval of the variable meta data in the future.

F. Code Generation

1) Mapping Keywords and Attributes to Code Constructs:

This mapping is done using Lex. Lex is used to identify patterns in the given text [32]. Different possibilities of keywords and attributes are specified for each code construct. Lex tries determining to which possibility the input text matches. The different possibilities are specified using regular expressions.

2) Generate Appropriate Code:

Appropriate code for the pattern matched using Lex, is generated using Yacc. The keywords and attributes are specified in the form of grammar. Yacc reads the grammar and generates the corresponding code [32].

IV. System Implementation

The proposed system is implemented using a Flask server as backend and a HTML webpage as front end.

A. Front End

1) Input: Users can either speak or type in natural language commands. The speech recognition module used is the JavaScript Web Speech API [33].

Algorithm 1: Speech Recognition

```
output: transcript:Recognized Speech Transcript
Function webSpeechRecognition():
    recognition = new webkitSpeechRecognition()
    recognition.continuous = true
    recognition.lang = 'language option'
    recognition.onresult = getTranscript(event)
Function getTranscript(event):
    for i ←event.resultIndex to event.results.length do
        if event.results[i].isFinal then
            print event.results[i][0].transcript
    end
end
```

Algorithm 1 shows the webSpeechRecognition object being initialized. Once language is set, recognition is started. When recognition is set as continuous recognition is done even when the user pauses. The onResult event handler gets the transcript after recognition. The web speech API is event based [34]. This helps in processing the speech to text asynchronously. Intermediate speech recognition helps give real time feedback back to the user.

Users can also directly type in natural language commands in the input text box. Once the user enters the command, the command is then sent to the Flask server.

2) Code Editor: The output after processing is displayed on a box which is the code editor. As well as displaying the output code, the editor also shows the line number, the cursor position and the number of characters in each line. The user can perform the following actions in the code editor:

a) Goto Line Number

b) Shift Cursor Right

c) Shift Cursor Left

These functions help ensure the output code is added in the correct position.

B. Lexical Analysis

Lexical analysis of the input command is done. The following steps are followed:

1) The request is checked whether it is for "space", “open braces", "closed braces", "enter", "semicolon". These are all basic punctuations needed in C for structure.

2) Next numbers are handled. The input command may have the text representation of numbers instead of the actual number. Word2number [35] package of python handles this.

3) Stop words are removed from the input command. Any word that should be ignored is removed.

4) Keywords of C are maintained in a separate file. The input command is checked to see if a keyword is present or not. Without a keyword the relevant code structure cannot be created and the system would throw an internal server error.

C. Syntax Analysis

First the input command is tokenized. Then parts of speech tagging are done using Averaged Perceptron Tagger [36] of NLTK. A feature list is created based on the tags. Attributes such as variable names, numbers, operations etc. are determined using POS tags. Once the feature list is generated, the feature list is sent to the Lex file of the determined keyword.

D. Semantic Analysis

Symbol tables are used to keep a track of variables and their types. Whenever a user specifies a variable name, type checking is done using the symbol table.

E. Code Generation

Each Lex file has various possibilities in which a sentence can be constructed for the particular keyword. The feature list is matched with one of the possibilities using Lex. Yacc generates a code structure for every corresponding Lex match.

V. Result and Analysis

The results obtained from the implementation are discussed below with their analyses. A sample set of commands for finding the factorial of a number is given below with its corresponding code in Fig. 6. Further, the analysis is done on the processing of the NL commands on the system.

```
1  int i                                          1  int i=0;
2  enter                                          2  unsigned int factorial=0;
3  unsigned int factorial                         3  printf("enter integer");
4  enter                                          4  int n=0;
5  print "enter integer"                          5  scanf("%d",n);
6  enter                                          6  if(n < 0)
7  int n                                          7  printf("error");
8  enter                                          8  else
9  scan n                                         9  for(i=1;i<=n;++i)
10 enter                                          10 factorial=factorial*i
11 if n lesser than 0                             11 printf("%d",factorial);
12 print "error"                                  12 end
13 enter
14 else
15 for loop
16 assign 1 to i
17 compare i lesser than equal to n
18 pre increment i
19 assign multiply factorial and i to factorial
20 print factorial
21 end
```

Fig. 6 Input Commands and the Corresponding C Code Output for a Simple Factorial Program

A. Time Since Request in HTTP for compiler operations

Fig. 7 shows the time for each natural language instruction to be processed and the corresponding C code to be generated. The maximum time taken for processing is 0.329 seconds. This shows that the processing time for Lex and Yacc to generate code is low. Hence the proposed system can be used in real time.

B. Accuracy of NL commands in Compiler

Constructs with higher accuracy shows that more different possibilities of sentence structures are handled by the Lex files. For lower accuracies it shows that the user has to stick to a certain structure when issuing the commands. Fig. 8 shows the accuracy of some sample operations performed in the system with natural language commands. It is observed that the looping statements have an accuracy around 82% to 96% as they can be expressed in semantically diverse forms. The different operations have shown high accuracy prompting the system to include and recognize diverse statements.

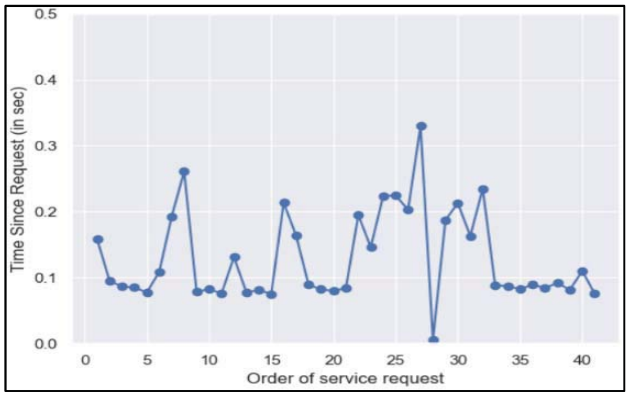

Fig.7 The Curve Plotted with Time Since Request in HTTP Connection for Different Instances

C. *Precision of NL Commands Measured for Each Type of Operation*

Precision is the proportion of correctly generated code to all code that is generated.

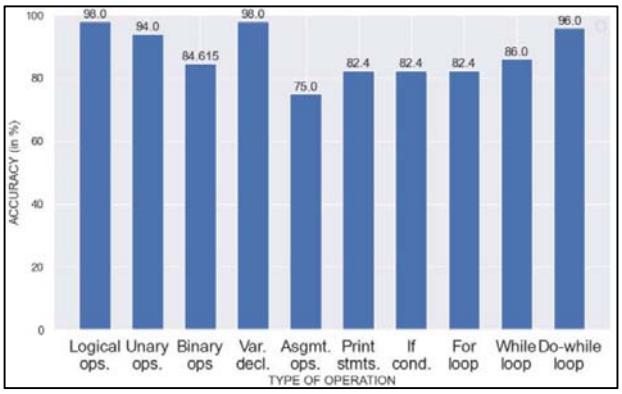

Fig. 8 Accuracy Obtained with Various NL Commands on The Compiler

$$\text{Precision} = TP/(TP+FP) \qquad (1)$$

where FP is the number of mistranslated code sequences. From Fig. 9, we observe that there is high percentage of precision with low chance of mistranslated code for NL commands. Mistranslated codes lead to semantic errors which are hard to diagnose while carrying out operations. This is avoided in the system by feeding the system with substantial keywords for recognition.

D. *Measure of Recall of Each NL Command for Each Type of Operation*

Recall is the proportion of successfully generated code to the number of input commands provided.

$$\text{Recall} = TP/(TP+FN) \qquad (2)$$

where FN is the number of input commands that are not converted to code. The recall values in Fig. 10 were calculated from the ratio of untranslated to translated words. It is seen that most constructs are translated completely. For constructs which are not generated, more rules should be added for code generation for a smaller number of attempts required for successfully generating the codes.

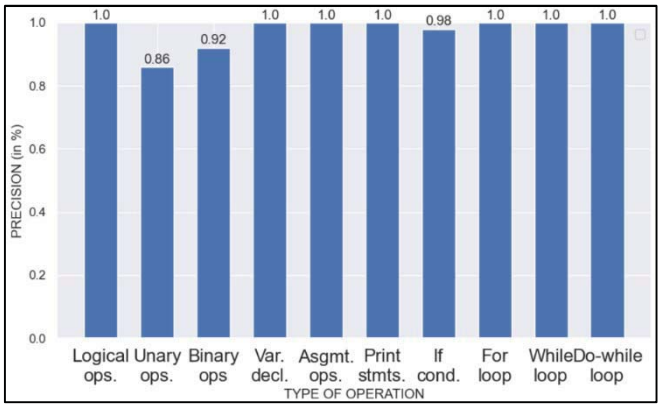

Fig.9 Precision of NL Command for Each Type of Operation

E. *Error Percentage in Characters with Each Construct*

The error percentage of characters is calculated by finding the number of missing or unnecessary characters or words for specifying a construct using NL. These values are tabulated in Table I. These errors are also handled by the compiler which displays message for correcting the words of NL command for optimum code conversion.

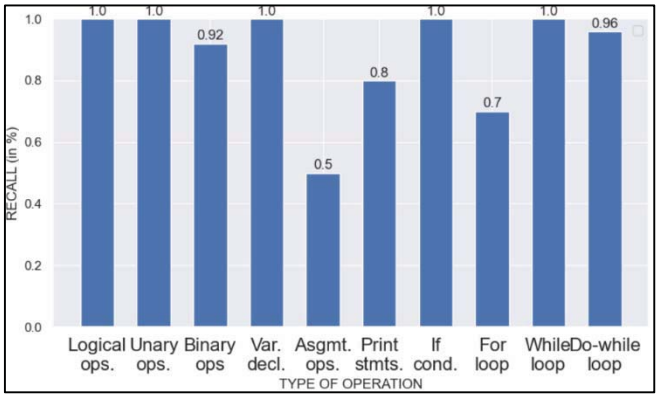

Fig.10 Recall of NL Command for Each Type of Operation

TABLE I. CHARACTER ERROR PERCENTAGE IN CONSTRUCTS

Construct	Error Percentage (%)
Unary Operations	21.05
Binary Operations	29.03
Assignment Operations	31.57
Print statements	55.31
If conditional	13.4
For loop	12.94
Do-while loop	9.52

VI. CONCLUSION AND FUTURE WORKS

The system built for conversion of speech and text recognition with natural language to C programming language is implemented. The system could be extended to assist in simplifying the complex interfaces [4]. The implementation of a code generator for natural language to an OOPS programming language such as Java or C++ is proposed as future work. For OOPS programming, handling of multiple classes and nesting of classes could be implemented with good performing accuracy to handle real-world problems [8]. Further, parsing could be implemented with dynamic learning [37] algorithms with neural mapping between NL commands and code snippets [38].

ACKNOWLEDGEMENT

We thank our Professor Dr. V. Mary Anita Rajam, Professor, College of Engineering Guindy, Anna University, India for her invaluable contribution to our project.

REFERENCES

[1] M. C. Surabhi, "Natural language processing future," 2013 International Conference on Optical Imaging Sensor and Security (ICOSS), Coimbatore, pp. 1-3, 2013

[2] P. M. ee, S. Santra, S. Bhowmick, A. Paul, P. Chatterjee and A. Deyasi, "Development of GUI for Text-to-Speech Recognition using Natural Language Processing," 2018 2nd International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), Kolkata, pp. 1-4, 2018

[3] T. Patten and P. Jacobs, "Natural-language processing," in IEEE Expert, vol. 9, no. 1, pp. 35-, Feb. 1994

[4] X. Liu and D. Wu, "From natural language to programming language". In Innovative Methods, User-Friendly Tools, Coding, and Design Approaches in People-Oriented Programming ,pp. 110-130, IGI Global, 2018

[5] A. Begel, "Programming by voice: A domain-specific application of speech recognition", In: AVIOS Speech Technology Symposium - SpeechTek West, February 2005.

[6] G. Campagna, S. Xu, M. Moradshahi, R. Socher, and M. S. Lam, "Genie: A Generator of Natural Language Semantic Parsers for Virtual Assistant Commands. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation" In: PLDI 2019, page 394410, New York, NY, USA, Association for Computing Machinery, 2019

[7] S. Shafiee, F. Almasganj, A. Jafari, "Speech/non-speech segments detection based on chaotic and prosodic features". In: Proc. Interspeech, pp. 111-114, 2008.

[8] S. Mandal and S. K. Naskar, "Natural language programming with automatic code generation towards solving addition-subtraction word problems", in Proc. of the 14th Int. Conf. on Natural Language Processing (ICON-2017) (NLP Association of India,Kolkata, India, December 2017), pp. 146–154, 2017

[9] C. Raju, M. Thirupathi and T. Arvind, "Analysis of Parsing Techniques and Survey on Compiler Applications", International Journal of Computer Science and Mobile Computing (IJCSMC), 2(10): 115-125, ISSN 2320-088X, 2013

[10] C. Northwood, *Lexical and Syntax Analysis of Programming Languages*. Accessed on: June 11, 2020. [Online]. Available: https://www.pling.org.uk/cs/lsa.html

[11] M.E. Lesk and E. Schmidt, *Lex – A Lexical Analyzer Generator*, Dinosaur Compiler Tools. Accessed on: June 11, 2020. [Online]. Available: http://dinosaur.compilertools.net/lex/index.html

[12] *Natural Language Toolkit*. (Version 3.5), NLTK. Accessed on: June 11, 2020. [Online]. Available: https://www.nltk.org/

[13] A. Singh, *"Introduction to YACC"*, GeeksforGeeks. Accessed on: June 11, 2020. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-yacc/

[14] Wikipedia, *C (Programming Language)*, 9 June, 2020. Accessed on: June 11, 2020. [Online]. Available: https://en.wikipedia.org/wiki/C_(programming_language)

[15] Tutorials Point, *C Programming Language*. Accessed on: June 11, 2020. [Online]. Available: https://www.tutorialspoint.com/cprogramming/

[16] A. Imam and A. Alnsour, "The Use of Natural Language Processing Approach for Converting Pseudo Code to C# Code" In: Journal of Intelligent Systems, 29(1), 1388-1407, 2019

[17] K. M. T. Rahit, R. H. Nabil and M. H. Huq, "Machine Translation from Natural Language to Code Using Long-Short Term Memory", In: Arai K., Bhatia R., Kapoor S. (eds) Proceedings of the Future Technologies Conference (FTC) 2019. FTC 2019. Advances in Intelligent Systems and Computing, vol 1069. Springer, Cham, 2020

[18] A. B. Kulkarni, S. S. Karandikar, P. A. Bamhore, S. R. Gawade and D. V. Medhane, "Computational Intelligence Model for Code Generation from Natural Language Problem Statement," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, pp. 1-6, 2018

[19] D. Price, E. Riloff, J. Zachary and B. Harvey, "NaturalJava: A Natural Language Interface for Programming in Java", Proceedings of the 5th International Conference on Intelligent User Interfaces ser. IUI '00, pp. 207-211, 2000.

[20] A. Trifan, M. Angheluș and R. Constantinescu, "Natural language processing model compiling natural language into byte code," 2017 International Conference on Speech Technology and Human-Computer Dialogue (SpeD), Bucharest, pp. 1-6, 2017

[21] V. Schlegel, B. Lang, S. Handschuh and A. Freitas, "Vajra: step-by-step programming with natural language." In :Proceedings of the 24th International Conference on Intelligent User Interfaces, 2019

[22] G. W. Furnas, T. K. Landauer, L. M. Gomez and S. T. Dumais, "The Vocabulary Problem in Human-System Communication", In: Commun. ACM, 30, 964-971, 1987.

[23] S. Graf, T. Herbig, M. Buck and G. Schmidt, "Features for voice activity detection: a comparative analysis", In: EURASIP J. Adv. Signal Process. Issue :2015, Article: 91, 2015

[24] A. V. Haridas, R. Marimuthu, V. G. Sivakumar, "A critical review and analysis on techniques of speech recognition: The road ahead", In: International Journal of Knowledge-Based and Intelligent Engineering Systems 22, pp. 39–57, 2018

[25] D. O'Shaughnessy, "Automatic speech recognition," 2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), Santiago, pp. 417-424, 2015

[26] D. Asenov and P. Muller, "Envision: A fast and flexible visual code editor with fluid interactions (Overview)," 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Melbourne, VIC,pp. 9-12, 2014

[27] T. Pay, "Totally automated keyword extraction," 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, pp. 3859-3863, 2016

[28] J. Kaur and P. K. Buttar, "A systematic Review on Stopword Removal Algorithms", International Journal on Future Revolution in Computer Science & Communication Engineering, pp. 207-210, Apr 2018

[29] S. G. Kanakaraddi and S. S. Nandyal, "Survey on Parts of Speech Tagger Techniques," 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), Coimbatore, pp. 1-6, 2018

[30] Y. Lee, "Design and Implementation of the Symbol Table for Object Oriented Programming Language", International Journal of Database Theory and Application Vol.10, No.7, pp.27-40, 2017

[31] Y. Son, Y. Lee, "The Semantic Analysis Using Tree Transformation on the Objective-C Compiler", In: Kim T. et al. (eds) Multimedia, Computer Graphics and Broadcasting. MulGraB 2011. Communications in Computer and Information Science, vol 262. Springer, Berlin, Heidelberg,2011

[32] M. Upadhyaya, "Simple calculator compiler using Lex and YACC," 2011 3rd International Conference on Electronics Computer Technology, Kanyakumari, pp. 182-187, 2011

[33] Mozilla Developers, *Using the Web Speech API*, May 17, 2020. Accessed on: June 11, 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API

[34] Julius Adorf, "Web Speech API" in KTH Royal Institute of Technology, Stockholm, May 2013.

[35] *word2number*. (Version 1.1), Python. Accessed on: June 11, 2020. [Online]. Available: https://pypi.org/project/word2number/

[36] *nltk.tag package*. (Version 3.5), NLTK. Accessed on: June 11, 2020. [Online]. Available: https://www.nltk.org/api/nltk.tag.html

[37] M. Atzeni and M. Atzori, "Translating Natural Language to Code: An Unsupervised Ontology-Based Approach," 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), Laguna Hills, CA,pp. 1-8, 2018

[38] U.Kusupati and V. R. T. Ailavarapu, "Natural Language to Code Using Transformers", https://www.cs.utexas.edu/~uday/data/deepcode.pdf [Online]