



Dissertation on
**Speak Pseudocode2c : A framework to convert
customised pseudocode to c code**

Submitted in partial fulfilment of the requirements for the award of degree of

**Bachelor of Technology
in
Computer Science & Engineering**

UE18CS390B – Capstone Project Phase - 2

Submitted by:

**Srishti Sachan
Rajdeep Sengupta
Raghav Aggarwal
Shaashwat Jain**

**PES1201802126
PES1201800144
PES1201800312
PES1201802346**

Under the guidance of

Prof. Nitin V Pujari
Dean-IQAC
PES University

June - December 2021

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING**

PES University

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru-560085, Karnataka, India



PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru-560085, Karnataka, India

CERTIFICATE

This is to certify that the dissertation entitled

**Speak Pseudocode2c : A framework to convert customised pseudocode
to c code**

is a bonafide work carried out by

**Srishti Sachan
Rajdeep Sengupta
Raghav Aggarwal
Shaashwat Jain**

**PES1201802126
PES1201800144
PES1201800312
PES1201802346**

in partial fulfilment for the completion of seventh semester Capstone Project Phase - 2 (UE18CS390B) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June - December 2021. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work.

Signature
Prof. Nitin V Pujari
Dean-IQAC

Signature
Dr. Shylaja S S
Chairperson

Signature
Dr. B K Keshavan
Dean of Faculty

External Viva

Name of the Examiners

Signature with Date

1. _____

2. _____

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **Speak Pseudocode2c : A framework to convert customised pseudocode to c code** has been carried out by us under the guidance of Prof. Nitin V Pujari, Dean-IQAC and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology in Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester June - December 2021. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1201802126 Srishti Sachan

PES1201800144 Rajdeep Sengupta

PES1201800312 Raghav Aggarwal

PES1201802346 Shaashwat Jain

Acknowledgements

I would like to express my gratitude to Prof. Nitin V Pujari, Department of Computer Science and Engineering, PES University, for his continuous guidance, assistance, and encouragement throughout the development of this UE18CS390B - Capstone Project Phase – 2.

I am grateful to the project coordinator, Prof. Silviya Nancy J, for organizing, managing, and helping with the entire process.

I take this opportunity to thank Dr. Shylaja S. S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B. K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

Abstract

Currently, we are surrounded by a world dependent on AI-based speech assistants which are built to make many trivial tasks easy. As the government aims to strengthen India's competitive edge in terms of IT, they have introduced coding as part of the main course curriculum from Class-VI onwards. At the moment, there is marginal support provided to help these students, to make them understand the programming concepts better. We propose a framework that retrieves text data from streamed voice input given by user and vice-versa, converts the raw text input to processed natural language formatted output pseudocode. Pseudocode makes an algorithm readable to students because it abstracts the details of programming languages and also lowers the language barrier faced by the students in learning how to define algorithms. Then, it maps the generated customised pseudocode to finished c code and well-defined algorithms to make programming easy for people who are new to the programming world in general and school students in particular.

Contents

List of Figures	6
List of Tables	6
1 INTRODUCTION	7
2 PROBLEM STATEMENT	8
3 LITERATURE SURVEY	9
3.1 [1] Improving Speech Recognition using Dynamic Multi-Pipeline API	9
3.1.1 Microsoft API	9
3.1.2 Google Cloud Speech API	9
3.1.3 Dynamic Multi-Pipeline API(DMP)	10
3.1.4 Results	10
3.2 [2] Automatic Translation from Pseudocode to Source Code: A Conceptual-Metamodel Approach	11
3.2.1 Approach	11
3.3 [3] Neural Named Entity Recognition from Subwords Units	14
3.3.1 Introduction	14
3.3.2 Model	14
3.3.3 Experimental analysis	15
3.3.4 Dataset	15
3.3.5 Experimental Findings	15
3.3.6 Conclusion	16
3.4 [4] The Use of Natural Language Processing Approach for Converting Pseudo Code to C# Code	16
3.5 [5] An XML-based Pseudo-code Online Editing and Conversion System	18
4 PROJECT REQUIREMENTS SPECIFICATION	21

4.1	Introduction	21
4.1.1	Project Scope	21
4.2	Product Perspective	21
4.2.1	Product Features	22
4.2.2	Operating Environment	22
4.2.2.1	Hardware Requirements	22
4.2.2.2	Operating System	22
4.2.2.3	Software Components	23
4.2.3	General Constraints, Assumptions and Dependencies	23
4.2.3.1	Assumptions and Dependencies	23
4.2.3.2	General Constraints	23
4.2.4	Risks	23
4.3	Functional Requirements	24
4.3.1	External Interface Requirements	24
4.3.1.1	User Interfaces	24
4.3.1.2	Hardware Requirements	25
4.3.2	Software Requirements	25
4.3.2.1	Code Editor(Inbuilt in GUI)	25
4.3.2.2	Google Cloud Platform	25
4.3.2.3	GCC compiler	26
4.3.3	Software Requirements	26
4.3.3.1	Communication between the Google Cloud API and frontend of framework	26
4.4	Non-Functional Requirements	26
4.4.1	Performance Requirement	26
4.4.2	Safety Requirement	26
4.4.3	Security Requirement	27
5	HIGH-LEVEL DESIGN	28
5.1	Introduction	28
5.1.1	Need for High Level Design Document (HLD)	28
5.1.2	Scope	28

5.1.3	Overview	28
5.2	Design Considerations	28
5.2.1	Design Goals	29
5.2.2	Architecture Choices	29
5.2.3	Constraints, Assumptions and Dependencies	31
5.2.3.1	Constraints And Dependencies	31
5.2.3.2	Assumptions	31
5.3	High Level System Design	31
5.3.1	Design Description	32
5.3.1.1	Master Class Diagram	33
5.3.1.2	Reusability Consideration	33
5.3.1.3	Swimlane Diagram	34
5.3.1.4	User Interface Diagram	34
5.3.1.5	External Interfaces	34
5.3.1.6	Packaging and Deployment Diagram	35
5.3.2	Help	35
5.3.3	Design Details	36
5.3.3.1	Novelty	36
5.3.3.2	Innovativeness	36
5.3.3.3	Interoperability	37
5.3.3.4	Performance	37
5.3.3.5	Security	37
5.3.3.6	Reliability	37
5.3.3.7	Maintainability	37
5.3.3.8	Portability	37
5.3.3.9	Legacy to modernization	38
5.3.3.10	Reusability	38
5.3.3.11	Application compatibility	38
5.3.3.12	Resource utilization, Etc.	38
6	LOW-LEVEL DESIGN	39
6.1	Introduction	39

6.1.1	Overview	39
6.1.2	Purpose	39
6.1.3	Scope	39
6.2	Design Constraints, Assumptions, and Dependencies	39
6.2.1	Constraints And Dependencies	39
6.2.2	Assumptions	40
6.3	Design Description	40
6.3.1	Master Class Diagram	41
6.3.2	Voice Input to Speech Module	41
6.3.2.1	Description	41
6.3.3	Use Case Diagram	42
6.3.4	Class Description - Mapper	42
6.3.4.1	Description	42
6.3.4.2	Data members	43
6.3.4.3	start_the_program	43
6.3.4.4	add_headers	43
6.3.4.5	add_main	43
6.3.4.6	declare_variable	43
6.3.4.7	initialize_variable	44
6.3.4.8	input_variable	44
6.3.4.9	assign_variable	44
6.3.4.10	print_variables	45
6.3.4.11	continued_if	45
6.3.4.12	while_loop	45
6.3.4.13	for_loop	46
6.3.4.14	end_func	46
6.3.4.15	get_program_list	46
6.3.4.16	comment	47
6.3.4.17	break_stmt	47
6.3.4.18	continue_stmt	47

6.3.4.19	process_input	47
6.3.5	Class Description - Pseudocode2c	48
6.3.5.1	Description	48
6.3.5.2	Data members	48
6.3.5.3	callback	48
6.3.5.4	run	48
6.3.5.5	save_code	49
6.3.5.6	compile_program	49
6.3.5.7	remove_junk	49
6.3.5.8	exit_code	49
6.3.5.9	insert_lhs	49
6.3.5.10	insert_rhs	50
6.3.5.11	show_alert	50
6.3.6	Class Description - MicrophoneStream	50
6.3.6.1	Description	50
6.3.6.2	Data members	50
6.3.6.3	__enter__	51
6.3.6.4	__exit__	51
6.3.6.5	_fill_buffer	51
6.3.6.6	generator	51
6.3.6.7	listen_print_loop	51
6.4	Sequence Diagram	52
6.5	Packaging and Deployment Diagrams	53
7	Proposed Methodology	54
7.1	Algorithm and Pseudocode	54
7.2	Implementation and Results	54
7.3	Further Exploration Plans and Timelines (optional)	54
	References	55

List of Figures

Figure 3.2(a)	The internal structure of an LSTM unit	12
Figure 3.2(b)	Sample output from CODE2PSEUDOCODE model	13
Figure 3.3(c)	LSTM implementation that was adopted by Lample	15
Figure 3.4(d)	Excel File Showing the Structure of the Linguistic Matrix	17
Figure 3.4(e)	The Output of Testing a Pseudocode Using CodeComposer	18
Figure 3.5(f)	Description of XML document by DOM model	19
Figure 5.2(a)	Comparison of Different Cloud Vendor	30
Figure 5.3(b)	Flow Chart for High Level Design	32
Figure 5.3(c)	Master Class Diagram	33
Figure 5.3(d)	Activity Diagram	34
Figure 5.3(e)	Activity Diagram	34
Figure 5.3(f)	Deployment Diagram	35
Figure 6.3(a)	Master Class Diagram	41
Figure 6.3(b)	Use Case Diagram	42
Figure 6.4(c)	Sequence Diagram	52
Figure 6.5(d)	Deployment Diagram	53

List of Tables

Table 3.2(a)	Percentage values of BLEU score for our CODE2PSEUDOCODE and the two techniques proposed in: Phrase-Based Machine translation (PBMT) and Tree-To-String Machine Translation (T2SMT).	13
Table 6.3(b)	Data Members in Mapper class	43
Table 6.3(c)	Data members in Pseudocode2c class	48
Table 6.3(d)	Table to show data members in Mapper class.	50

Chapter 1

INTRODUCTION

Currently, we are surrounded by a world dependent on AI-based speech assistants which are built to make many trivial tasks in our lives easy. For most of us, the ultimate luxury would be an assistant who always listens for your call, anticipates your every need, and takes action when necessary. That luxury is now available thanks to artificial intelligence assistants, aka voice assistants. Even these voice assistants are implemented through millions of lines of code. Today, coding is being taught from primary levels at school. According to the new education policy, programming has been introduced as a compulsory course for students from junior classes.

At the moment, there is very less support provided to help these students understand the programming concepts better. With the surge in the number of programmers and advancements in technology, the increasing competition among students can be very intimidating for a child. Also, the c code given in textbooks is empirically observed to be not very easy for students to comprehend at the school level when compared to natural language pseudocode. Editing huge lines of unformatted code can sometimes be frustrating even for a professional programmer. Therefore, our goal is to use a voice-based digital assistant to automate the process of writing and editing code by providing our voice input.

Our aim for this project is to make these children familiar with the coding environment so that they do not shy away from programming and they have an all time accessible tool to help them in their programming journey. This tool will take a user's voice input as pseudocode and convert it into a c program, given that the user gives the correct pseudocode as voice input.

Chapter 2

PROBLEM STATEMENT

Speak Pseudocode2c: A framework to convert customized pseudocode to c code. Today, coding is introduced in the early stages of school to prepare them better in the programming world. But this alone is not enough for students to get a good grip on programming. Introducing coding at an early age comes with its own challenges. These challenges include getting familiar with a totally different language when they are also coping with other subjects in their school curriculum, not having a proper platform to help them with coding i.e not getting enough practical experience with coding. Our tool will help a student with the experimental aspect of coding. Writing and implementing huge lines of code can be a great deal of hassle for a coder. Doing everything manually can be a real pain sometimes. This tool can help programmers with editing their code just by providing their voice input. Manual labor will be decreased to a great extent.

Chapter 3

LITERATURE SURVEY

3.1 [1] Improving Speech Recognition using Dynamic Multi-Pipeline API

We all use speech recognition in our daily lives. However, the problem with these is that sometimes they don't respond due to some fault in the input taken or some misunderstanding in the meaning of speech. Many studies are going on to minimize the errors. These errors can be caused due to incorrect meaning, substituted words or some missing phrases in the input taken.

This study proposes the method to address the problem of choosing between different speech models. It combines 2 stages using Microsoft API and Google API model voice.

Recent research shows that the most popular operating systems like Android and Microsoft so Google and Microsoft API are used in this research.

3.1.1 Microsoft API

Windows Speech API is used widely across the globe for speech to text conversion. All the microsoft windows systems use this speech API integrated inside windows Cortana. This application interface can be adapted as a toolbox which allows users to implement speech recognition applications.

Microsoft has improved its engines with deep neural networks and hidden markov models. This enables long speech recognition.

3.1.2 Google Cloud Speech API

The main advantage of Google Cloud API is the ability to support real time speech to text conversion. It also has multiple language support. It is implemented using deep neural networks. The error rate has reduced from 23% in 2013 to 8% in 2015. Speech recognition is generally measured using Word Error Rate(WER). Some popular cloud speech APIs have the following Word Error Rate:

- Google API: 9%
- Microsoft API: 18%
- CMU Sphinx: 37%

Some studies have shown that when the audio is recorded using a microphone in the form of stereo, the error rate drastically reduces to 3%.

3.1.3 Dynamic Multi-Pipeline API(DMP)

The speech is generally taken input through a microphone in .wav format and has a frequency of 16,000Hz. The input speech file is transformed to characters and phrases, called phonemes, using speech to text engines.

There is a detection value defined between 0 to 1 where 1 means detectable speech and 0 is undetectable. If the value is less than 0.5, then the speech will be passed through Google API and then the detection value is recalculated.

The mechanism behind this is that Microsoft API will add words and retrieve variables to compare words and speech. Whereas Google API will recode the speech and transform to WAV 1,600Hz using a wave library. Later on, the speech recognition happens via Google cloud speech API and it returns JSON string

3.1.4 Results

Each speech recognition API uses different processing algorithms so it is best to combine their abilities in order to get the best out of them. For the testing stage, Word Error Rate was calculated. The results are as follows:

- Google API: 11%
- Microsoft API: 14%
- Multi Pipeline Combination: 6%

The recognition accuracy for 70dB sound are:

- Microsoft API: 65%
- Google API: 82%
- Multi Pipeline Combination: 86%

3.2 [2] Automatic Translation from Pseudocode to Source Code: A Conceptual-Metamodel Approach

Pseudocode is written in natural language along with mathematical expressions which are useful for describing the source code. Pseudocode eases the process of understanding the written code for programmers irrespective of the programming language. However, writing pseudocode for a particular source code is not an easy task as it takes extra time. The problem which the authors are trying to solve is to generate pseudocode from given source code automatically and to achieve this goal, they have built a neural network with the help of deep learning using Long Short-Term Memory architecture.

3.2.1 Approach

The input to code2pseudocode model is source code written in python language and the output is the natural language pseudocode with mathematical notations in it. Each line of code is tokenized i.e each statement is considered as a sequence of tokens. For example:

$$os.remove(fname) \longrightarrow os, ., remove, (, fname,)$$

The encoder and decoder are crucial components of the code2psuedocode model. Both of the components are Recurrent Neural Network (RNN) which has a number of Long Short-Term memory (LSTM) units. LSTM is a special neuron for reciting long-term dependencies. It has an internal state variable that is transferred from one cell to the next and is changed using operation Gates. It is intelligent enough to know how long to keep old records, what to recall and forget, and how to make connections between old memory and new knowledge.

For translation of long statements, instead of translating reading the complete sentence at once they translated one phrase at a time towards the completion of the sentence. An attention layer is applied to the model to decide the contribution of each token in the input chain, so that when forecasting a new output token, previous token contribution accounts for the prediction outcomes.

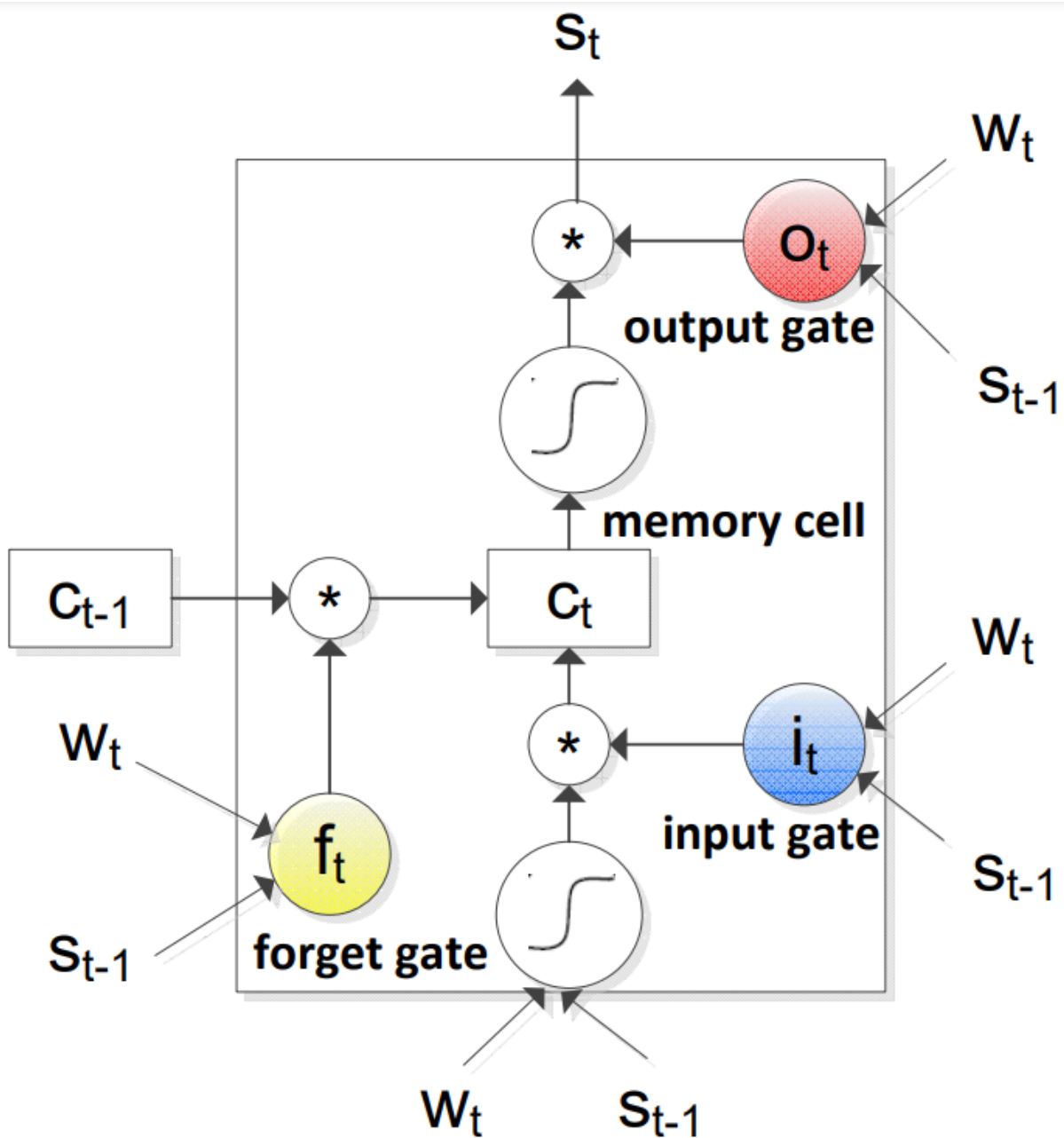


Figure 3.2(a): The internal structure of an LSTM unit

The authors use a technique called teacher-forcing technique. The Ideology behind teacher-forcing technique is: Consider a math exam question with several sections, of which the answer to part (a) is necessary for the equation in part (b), the answer to part (b) is required for part (c), and so on. To get the correct answer, we must pay extra attention to these questions because if part (a) is incorrect, all subsequent parts will be incorrect as well, despite the fact that the formulas and equations are correct. Similarly, Teacher Forcing takes care of this in the following manner: When we get an answer for part (a), a teacher can equate it to the right one, record the

score for part (a), and remind us what the correct answer is so we can use it for part (b).

This is a snippet of their sample result which they have added in their paper.

Python Code	Pseudo-Code
<u>Manually-Generated Pseudo-Code</u>	
def npgettext(context, singular, plural, number)	# Define the function "npgettext" with 4 arguments: context, singular, plural, and number.
del self._expire_info[key]	# Delete the value under key "key" of "self._expire_info".
return self.render()	# Call the method "self.render", return the result.
yield representative, True	# Yield representative and boolean True.
<u>Code2Pseudocode Encoder-Decoder</u>	
def npgettext(context, singular, plural, number)	# Define the function "npgettext" with 4 arguments: context, singular, plural, and number.
del self._expire_info[key]	# Delete the value under the key "key" of the "self._expire_info" dictionary.
return self.render()	# Call the method "self.render", return the result.
yield representative, True	# return is boolean True.

Figure 3.2(b): Sample output from CODE2PSEUDOCODE model

For benchmarking and comparing their result with other techniques, they used the same dataset as used by other authors implementing different techniques. The sample dataset which is used is retrieved from the code base of Django which is a framework for web application. They hired an engineer to extract it's source code which is written in python and create pseudocode for that particular source code. They obtained 18,800 pairs of code along with pseudocode for their dataset. To measure the quality of generated pseudocode they used a scale called Bilingual Evaluation Understudy (BLEU). BLEU is a technique to evaluate the quality of text when text is converted from one machine translated language to another natural language.

Pseudo-Code Generator	BLEU%
Code2Pseudocode	54.78
T2SMT	54.08
NoAtt	43.55
PBMT	25.17
SimpleRNN	06.45

Table 3.2(a): Percentage values of BLEU score for our CODE2PSEUDOCODE and the two techniques proposed in: Phrase-Based Machine translation (PBMT) and Tree-To-String Machine Translation (T2SMT).

The problem which we are trying to solve is the reverse of the problem defined above i.e converting pseudocode to source code but our approach is completely different because we are trying to build a framework which links various predefined functionalities/applications to achieve the goal. For taking pseudocode as input, we are giving facility to user to speak the pseudocode

(close to natural language) instead of typing it and the voice input will be displayed on the monitor by using the Google Cloud's Speech-To-Text API and that pseudocode will be processed and automatically converted into source code using our custom defined mapping library.

3.3 [3] Neural Named Entity Recognition from Sub-words Units

3.3.1 Introduction

In communication in language innovation applications, for example, voice-controlled brilliant partners like the Amazon Echo or Google Home, named element acknowledgment (NER) is a basic job. Its aim is to find instances of named entities in text.

As new expressions are gotten from the framework over the long run, NER models are retrained at ordinary spans. Word-level representations are used in current models, or they are combined with character-level representations.. Shortcomings of such models –

1. Large memory requirements and longer training time.
2. Combination with other subword units is ignored leading to low accuracy.
3. Out-of-vocabulary (OOV) words can pose a problem.

The neural solution used in this paper is based on subword units, such as letters, phonemes, and bytes. We take in portrayals from every one of the three subword units for each word in an expression.

3.3.2 Model

The model learns a low-dimensional representation from each subword unit (character-, phoneme-, and byte-level) for each word in an utterance, which is then concatenated and fed into a bidirectional LSTM-CRF model.. LSTM implementation used by the model–

$$\begin{aligned}i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\c_t &= (1 - i_t) \odot c_{t-1} + \\&\quad i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\h_t &= o_t \odot \tanh(c_t),\end{aligned}$$

Figure 3.3(c): LSTM implementation that was adopted by Lample

where W 's are shared weight grids, b 's are the inclinations, σ is the component astute sigmoid capacity, x_t addresses the token at position t , h_t is covered up state at time t .

A softmax layer is utilized on top of the yield of the bidirectional LSTM organization to figure a likelihood appropriation of yield labels for a given token.

3.3.3 Experimental analysis

1. To examine the yield of individual subword units and different blends of subword units, both with and without the utilization of word-level embeddings.
2. Compare the output of subword-only models, dedicated word-level embedding models, and models that combine both.
3. Look at whether subword units can help with out-of-vocabulary terms.

3.3.4 Dataset

A wide genuine world dataset that incorporates American English (EN), German (DE), French (FR), and Spanish (ES) (ES). The data depends on client demands for voice-enacted gadgets that were physically deciphered and clarified with named elements.

3.3.5 Experimental Findings

1. Different subword units provide the same results for different languages when used individually.

2. For all languages, results improve when many subword units are combined, and for all but French, the best results are obtained when all of the subword units are combined.
3. The best combination for French is characters and bytes.
4. Additive gains are acknowledged by using several subword units compared to using only one.

3.3.6 Conclusion

1. With larger training data, performance of models trained solely on subword units approaches that of models with dedicated word-level embeddings (91.35 vs 93.92 F1 for EN), but with smaller vocabulary size, performance of models trained solely on subword units falls short of that of models with dedicated word-level embeddings .
2. In the presence of out-of-vocabulary terms, the subword-based model outperformed the corresponding word-level model (34.9 vs 34.8), whereas the combined model achieved 37.1 F1, suggesting that subword units are useful.
3. Combining subword units improves performance.

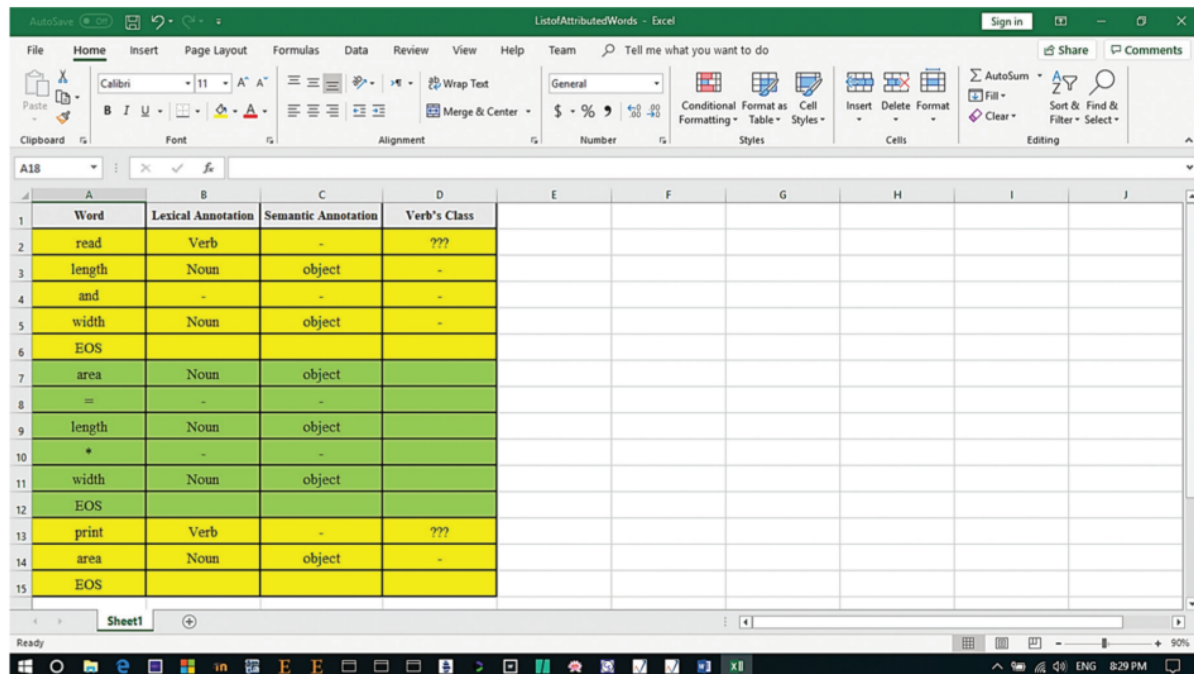
3.4 [4] The Use of Natural Language Processing Approach for Converting Pseudo Code to C# Code

The paper talks about a few techniques to enhance and make possible what the author calls the process to computerize the structure of a programming language code from pseudocode, which is seen here as an interpretation interaction for a characteristic language text, as pseudocode is an arranged book in normal English language.

The first tool it talks about is an Automatic Code Generator, abbreviated as ACG, which uses programs to generate source code which humans would otherwise have to write. They can be either passive or active. Passive code generators generate code that needs some modification by the user before running, whereas active code generators are embedded in the software development process.

The paper also talks about the use of Natural Language Processing (NLP) techniques to format the input in various ways to take multiple inputs/ slangs but generate the same source code which the user wants. This helps in clubbing various actions and making the natural language used in the approach to be more easy and not a fixed or particular pseudo code language.

The author proposes the “CodeComposer System”, which is a Machine Translation tool that uses Semantic Role Labelling(SRL) and NLP techniques to identify and convert pseudocode to mapped C # code.



	A	B	C	D	E	F	G	H	I	J
	Word	Lexical Annotation	Semantic Annotation	Verb's Class						
1	read	Verb	-	???						
2	length	Noun	object	-						
3	and	-	-	-						
4	width	Noun	object	-						
5	EOS									
6	area	Noun	object							
7	=	-	-							
8	length	Noun	object							
9	*	-	-							
10	width	Noun	object							
11	EOS									
12	print	Verb	-	???						
13	area	Noun	object	-						
14	EOS									

Figure 3.4(d): Excel File Showing the Structure of the Linguistic Matrix

It uses the semantic nature of the pseudocode’s language to find out the verbs and maps them into actions and the nouns and maps them into variables or mathematical equations to process on the system.

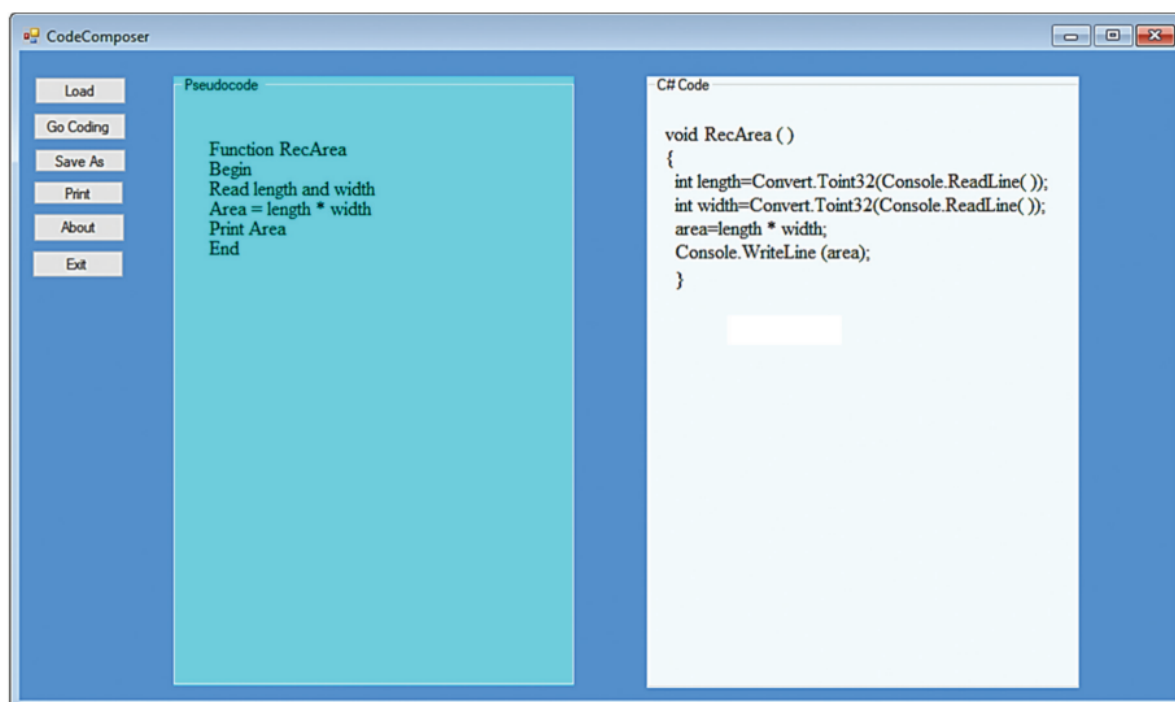


Figure 3.4(e): The Output of Testing a Pseudocode Using CodeComposer

An assessment of the exactness of CodeComposer utilizing a binomial method shows that it has an accuracy of 88%, a review of 91%, and a F-proportion of 89%.

3.5 [5] An XML-based Pseudo-code Online Editing and Conversion System

The paper introduces the approach of trying to convert and edit pseudo code to source code in multiple languages by using XML (eXtensible Markup Language) and the features of DOM(document object model) by encapsulating features of each line of code in some tags eg: `< var >`, `< var – type >`, `< var – name >`, `< var – value >` etc.

DOM is utilized as an article arranged information interface. DOM is an assortment of articles, which characterizes the document structure. What's more, developers can peruse, traverse, change, add and erase on XML archives through the control of these items. DOM can guarantee the right employments of punctuation and design, and improve on the procedure on the record, and furnish a decent blend with the information base change. DOM can make a tree of hubs (alluded to nodetree) in light of the data of XML archives. Also, we can utilize these interfaces and the tree design to get to the data of these XML records.

```
<?xml version="1.0" encoding="gb2312"?>
<code>
  <var>
    <varname> temp</varname>
    <vartype>int</vartype>
  </var>
  <array >
    <arrayname> str</arrayname>
    <arraytype>String</arraytype>
    <arraysize>4</arraysize>
  </array>
</code>
```

Expressed by DOM, as shown in Figure 1:

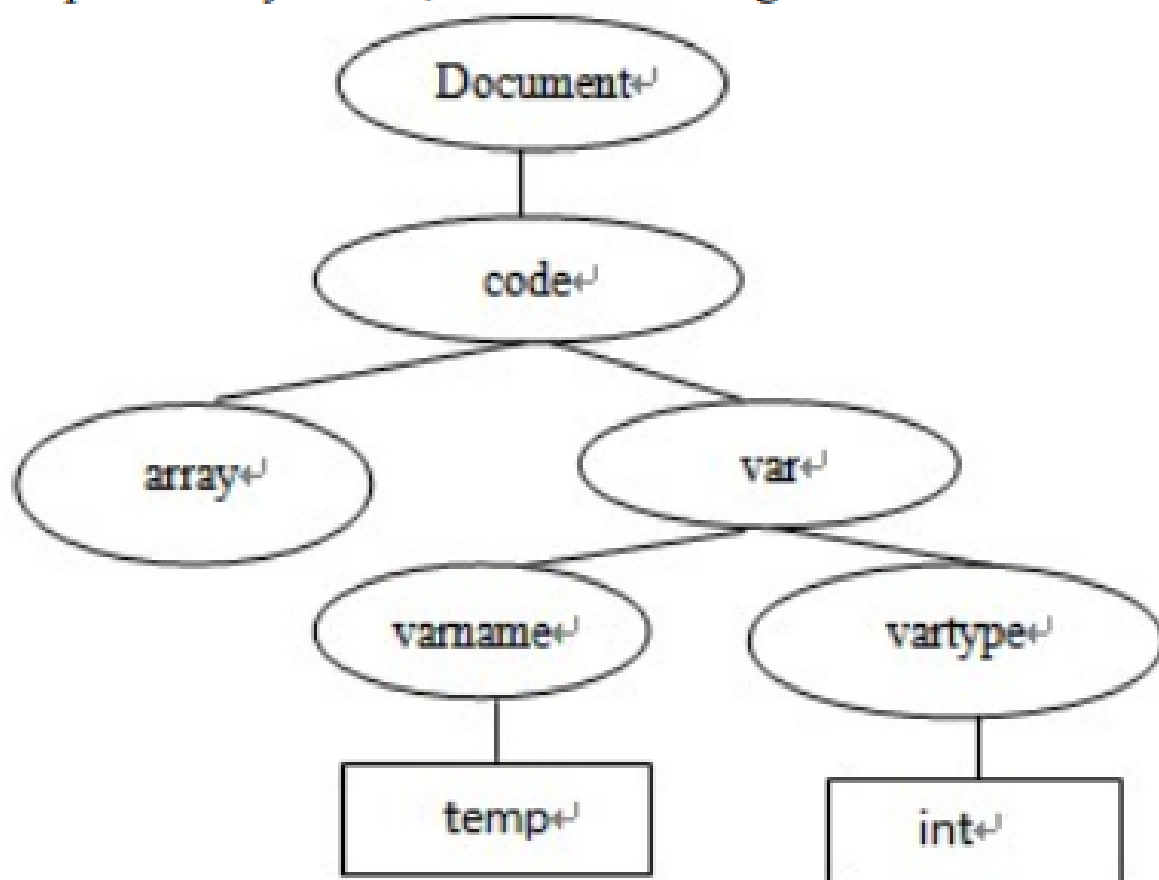


Figure 3.5(f): Description of XML document by DOM model

This model has the benefit of a systematic approach and the use of this converted document to be written in any language the user wishes and not just c code. The author mentions the use of `< cyc >`, `< while >`, `< for >`, `< do >` and various tags for loops, and for blocks where even nesting is important, it introduces the `<nesting>` tag which can encapsulate various statements.

Chapter 4

PROJECT REQUIREMENTS SPECIFICATION

4.1 Introduction

We are surrounded by a world dependent on AI-based speech assistants which are built to make many trivial tasks in our lives easy. Even these voice assistants are implemented through millions of lines of code.

According to the new education policy, programming has been introduced as a compulsory course for students from junior classes. At the moment, there is very less support provided to help these students understand the programming concepts better. The c code given in textbooks is empirically observed to be not very easy for students to comprehend at the school level when compared to natural language pseudocode.

Editing huge lines of unformatted code can sometimes be frustrating even for a professional programmer. Therefore, our goal is to use a voice-based digital assistant to automate the process of writing and editing code by providing our voice input.

4.1.1 Project Scope

The scope entails providing a framework for students and beginners to get familiarized with the concept of programming and not overwhelmed by the huge lines of code with hard rules. We plan on doing so by retrieving text data from streamed voice input given by user and vice-versa. Once we acquire this text, we convert raw text input to processed natural language formatted output. The natural language formatted output can also be seen as a pseudocode which we then use for the final stage i.e. mapping customised pseudocode to finished c code and well-defined algorithms.

The user can edit and execute the code in a similar fashion and be independent of the rules of typing for the language and focus on the algorithms more. The tool can be helpful for better understanding and can target developers and students at all levels in the future.

4.2 Product Perspective

Speak Pseudocode2c is a commercial product which will work as a cloud based application. It consists of three phases:-

Retrieving the text from cloud: The user accesses the microphone present at client end. This

input will be streamed and preprocessed at cloud platform (server end) and the user will get the output as text, with minimal errors and maximum accuracy, on the console window.

Formatting text into Tags: The important keyword is picked from received text will be translated into a defined formatted structure which will make it easy to map pseudo-code to source code.

Mapping the Tags to source code: The tags will act as an intermediate model between pseudo-code and source code (c program) which helps in mapping of the same.

4.2.1 Product Features

With the speak pseudocode2c the user only has to speak the basic pseudocode and the well-defined structured and formatted c program will be shown on the display screen.

The user doesn't have to press any key neither for writing the basic c program nor for compiling and executing the program. The only workload on the user is to speak the pseudocode.

Using Google Text-To-Speech API, available on Google Cloud Platform which is a cloud service provider, makes it easy to convert speech to text with high accuracy.

4.2.2 Operating Environment

4.2.2.1 Hardware Requirements

- The user needs a system which has at least a basic text editor and support for gcc compiler and python3.7. Using an IDE is a big plus.
- Microphone: For receiving the speech input the system should have an in-built microphone or external microphone which capture sound in good quality.
- Speakers: After the text is generated from voice input the framework will use Speech-to-Text and ask the user for confirmation whether the text is right or wrong by saying “yes” or “no”. For this there is a requirement for good quality speakers.

4.2.2.2 Operating System

- The framework can work on 32-bit and 64-bit Linux, Windows, as well as on Mac OS.
- The machine should have a minimum of 4GB RAM and 256 GB secondary memory.
- A good internet requirement is a key requirement for this framework to work.

4.2.2.3 Software Components

- GCC compiler need to be installed on the host machine.
- Python3.7 needs to be installed.
- The basic text editor for displaying the program on the screen.
- Google Cloud python library needs to be installed in order to run the framework.

4.2.3 General Constraints, Assumptions and Dependencies

4.2.3.1 Assumptions and Dependencies

- The user has good internet connectivity.
- Microphone and Speakers are working fine at the host machine.
- Errors from the cloud service provider.
- The client has all the software requirements installed on the machine.
- If the machine does not have enough hardware resources available for the application then the users might have to allocate them with other applications.

4.2.3.2 General Constraints

- Failure of microphone/speaker may stop the program.
- Unknown keywords picked up from the speech or unclear speech due to background noise.
- Incomplete or erroneous installation of the gcc compiler on the host machine.
- Different python3 version installed on the host machine.

4.2.4 Risks

The possible risk which might occur are: —

- Retrieving voice input from the cloud with subpar accuracies.
- Formatting the text after successfully retrieving it from the cloud.

- Creating a custom natural language-based dictionary from pseudocode for c language.
- Real-time updation of c code in the editor.
- Crashing of program because of different versions of python3 and gcc compiler.

4.3 Functional Requirements

As our goal is to convert spoken pseudocode to text and then map it to code, the main functional requirements include a user interface being a code editor where the user will be able to make changes to the code by using their voice. The program should be updated in real-time and visible to the user in the code editor as the user speaks. This might include some latency in the beginning as we are using Google Cloud Speech engine.

Our goals will include to implement this framework with minimal latency. This will be an interactive framework which confirms with the user whether to make any changes or to compile the final code and act accordingly.

4.3.1 External Interface Requirements

4.3.1.1 User Interfaces

- The framework will be accessible to the user through a software GUI built with native components from the system itself.
- We are planning on using a voice based system so the text interaction will be a choice for the user to opt from. For better readability, we'll go with textbook fonts like Arial, Cambria, Times New Roman.
- The layout will be available currently only for systems and not mobile. The layout will be horizontal with one half of the screen for the user and the other half for the code to be showcased for easy editing.
- The standard functions will include : language, editor, execute program, stop recognizing, licensing and help.
- The relative timing of inputs will depend on the user's voice input and his internet connection to the cloud. The only delay on the code side which will have to be optimised for being sub-second is the conversion of pseudocode to finished c code.

- The programmable functions will be the input microphone, language spoken and to display recognized text or not.
- The error messages will be displayed on the screen as dialog boxes with minimal UI and also be spoken out to the user for enhanced accessibility.

4.3.1.2 Hardware Requirements

- Microphone : The device types that can be supported are the default installed microphone inputs with properly installed drivers. The microphone will communicate through system permissions and no default protocol is needed.
- Speaker : A default output speaker is required although is optional in the case the user doesn't want any voice output throughout the program. The speaker will communicate through system permissions and no default protocol is needed.

4.3.2 Software Requirements

4.3.2.1 Code Editor(Inbuilt in GUI)

- Name and Description: GUI implemented by Tkinter
- Version / Release Number: 1.0
- Operating Systems: Across all platforms
- Tools and libraries: Tkinter in python

4.3.2.2 Google Cloud Platform

- Name and Description: Google Cloud Speech to Text
- Version / Release Number:
- Operating Systems: Windows 10, MacOS, Linux
- Tools and libraries: Text to Speech and Speech to Text

4.3.2.3 GCC compiler

- Name and Description: gcc compiler
- Version / Release Number: gcc version 9.3.0 or above
- Operating Systems: Windows 10, MacOS, Linux

4.3.3 Software Requirements

4.3.3.1 Communication between the Google Cloud API and frontend of framework

- Protocol: HTTP/HTTPS
- Client port: <any>
- API port: 80/443
- Allowed methods: GET, POST, PUT, DELETE, UPDATE

This communication between the client and the cloud platform is essential and the only communication for the whole framework. This communication is responsible for the speech to text conversion and the return methods will provide us with the text for further processing.

4.4 Non-Functional Requirements

4.4.1 Performance Requirement

- The reliability of the framework will be dependent on google cloud and user defined libraries for speech to text conversion.
- It can be accessed by anyone around the world.
- It is an on-demand service, hence requires no prior installations.
- It is operable only if the user has a stable internet connection.
- The tool will be compatible with all operating systems thus making it more flexible to use.

4.4.2 Safety Requirement

- The framework provides built-in protection for all user files as it is reliable on google cloud.

4.4.3 Security Requirement

- No authentication is required to use the product, only licensing of the software.
- Confidentiality and data integrity is provided as per google cloud services.
- A user's files cannot be accessed by other users accessing the tool.

Chapter 5

HIGH-LEVEL DESIGN

5.1 Introduction

5.1.1 Need for High Level Design Document (HLD)

This document is the high level design document for the “Speak Pseudocode2c”. The purpose of this High Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

5.1.2 Scope

The HLD documentation presents the structure of the framework, such as the master class diagram, system architecture, High Level Design, State Diagram (flow) and deployment model. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system. The goal is to display the code on screen via. speaking. The abstract is that the users will speak the natural language pseudocode in the microphone and corresponding to that pseudocode it's respective c language code will be generated. Mapping pseudocode to source code will be done automatically via the framework.

5.1.3 Overview

This document focuses on the general principles of design and building the framework for speak Pseudocode2c systems and explain the high level concepts of different layers in the framework. This will introduce the methodology used for the evaluation and further refinement of concepts, as well as acts as a guidance for the process of identifying the necessary technologies required and used.

5.2 Design Considerations

According to the new education policy by the government, coding is now mandatory at school level, the support for it is negligible. Speak Pseudocode2c framework will boost and enhance the coding experience for the students. The design concepts for Speak Pseudocode2c framework will have to take into account aspects related to installation, updation and maintenance. The

framework uses the Cloud services for processing the voice input. The advantages of it is as follows: –

5.2.1 Design Goals

- **Processing Speed:** The client machine has minimal processing to do because most of the processing will be done on the cloud server. The voice input will be streamed as it will be broken into chunks of data.
- **Accuracy:** Using the Cloud API such as Speech-To-Text provides a large corpus of words as well as support for different languages, which increases our output space for the words. As Google cloud Speech process engine is very superior to other cloud vendors, therefore the accuracy is very high (more than 90%).
- **Security/Privacy:** As the voice input is sent to the Google Cloud via. a json format key, which is unique for every user provided the google service account, hence providing security is again the responsibility of the Google cloud.
- **Availability:** Google cloud is known for the it's 24x7 availability. The framework just needs an active internet connection to connect to the cloud for sending and receiving the data in the required format.

These all qualities indicate and cover all the basic requirements which are necessary for any product that is launched in the market. Therefore, our approach for choosing the cloud to convert Speech-To-Text is better and more reliable than other methods in building Speak Pseudocode2c framework.

5.2.2 Architecture Choices

Our team tried different methods to convert Speech-To-Text which are as follows: Offline Mode:

- Neural engine
- Mozilla deep speech
- CMU Sphinx

The accuracy of converting Speech-To-Text offline was very low (approx. less than 50%). Most of the words aren't even getting recognized and processing time was high as all the processing is happening on the client machine. Accuracy is the primary why we opted for the cloud approach. Online Mode(Cloud):

- IBM Watson
- Amazon AWS
- Google Cloud
- Google Speech-to-Text (without cloud's API)

The low results of offline Speech-To-Text made us look for other options. Using cloud API for achieving the goal solved our problem in hand but different cloud vendors have different accuracy, therefore we compared different API as mentioned above. It turns out Google Cloud provides the most accuracy and speed for converting Speech-To-Text. The comparison is shown in Fig 1. For us the accuracy for Google Cloud Speech-To-Text was 93% and word error rate was less than 10%.

Therefore, using Google Speech-To-Text seems to be a good option and it's advantages are already described.

Comparison Table

	Amazon Transcribe	Google Cloud Speech-to-Text	IBM Watson Speech-to-Text	Nuance Dragon NaturallySpeaking
Languages	2	120	7	43
Multiple Speaker Identification	✓	✓	✓	✓
Audio Sample Rate	8kHz to 48 Kh	16 kHz	16 kHz	16 kHz
Custom Vocabulary	✓	✓	✓	✓
Timestamps	✓	✓	✓	✓
Confidence Score	✓	✓	✓	✓
Profanity Filter	✓	✓	✓	✓
Price	\$0.00056 per second	\$0.006 per 15 seconds	\$0.02 per minute	From \$150 lifetime deal

Figure 5.2(a): Comparison of Different Cloud Vendor

5.2.3 Constraints, Assumptions and Dependencies

5.2.3.1 Constraints And Dependencies

- Currently, we are using the Google Cloud free tier which restricts us from using their service for any commercial applications. We have to abide by their terms of service for the same.
- Google Cloud free tier has a limitation for Speech-to-Text API which we are using in the development phase of the application, it is free for 60 minutes/month afterwards it will use the credits. Compared to other cloud vendors' Speech-To-Text Google provides very less free availability of this API.
- The hardware requirements for the system are minimal, although there is a requirement for a microphone and speaker with a working internet connection.

5.2.3.2 Assumptions

- The user is familiar with the pseudocode of at least one programming language.
- User has a working internet connection.
- The host machine has a gcc compiler installed for execution of the c programs along with python3 and google cloud python library which is necessary for running the framework.
- There is no problem with User Microphone and Speakers.
- The Background noise is minimum where the client is using the application.

5.3 High Level System Design

- Logical user groups : Students in preliminary classes with coding as a subject, beginners in the world of coding, developers.
- Application component : Google Cloud API, Google Speech to Text API, Google Text to Speech API, Formatting identified text into pseudocode, mapping pseudocode to executable c code
- Data Component : Mapping dictionaries, verb corpus for NLP processing

- Interfacing systems : Google Cloud API and mapping algorithm with mainframe integration and UI.

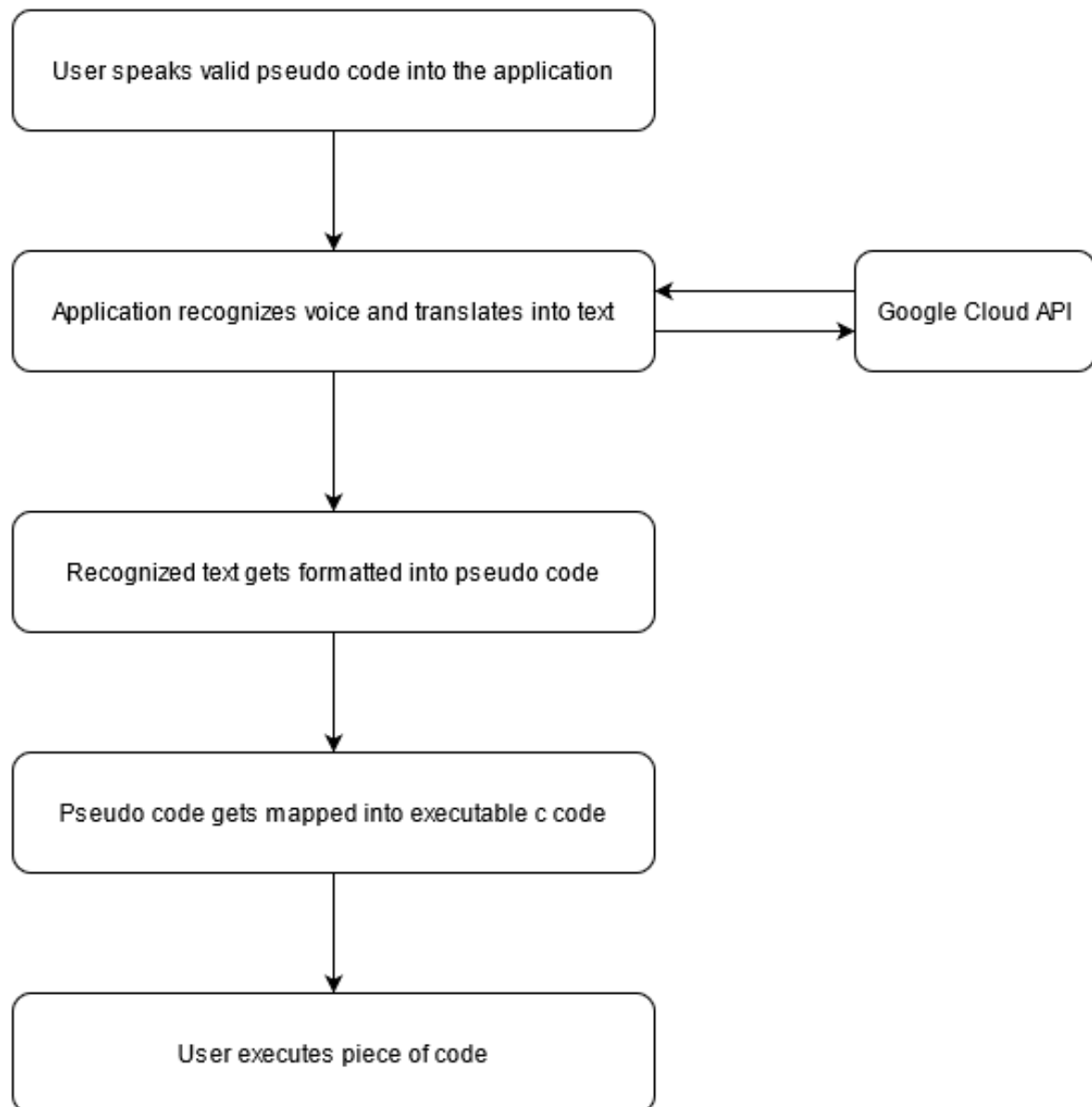


Figure 5.3(b): Flow Chart for High Level Design

5.3.1 Design Description

The main classes in the class diagrams include client/user, Google Cloud Platform, the terminal on which the program is run and the GUI of the program.

Initially the client has to register using their Google account to the framework. The client then starts speaking to the program. They speak the general pseudocode. The spoken pseudocode is sent to the Google Cloud Platform which returns the formatted pseudocode text. This is done

through API call along with many optimizations for voice to text conversion. The pseudocode text is returned to the terminal on which the program is run. Further, the pseudocode text is mapped to the C code. Finally, a GUI is created which contains the pseudocode and the mapped C code as output.

5.3.1.1 Master Class Diagram

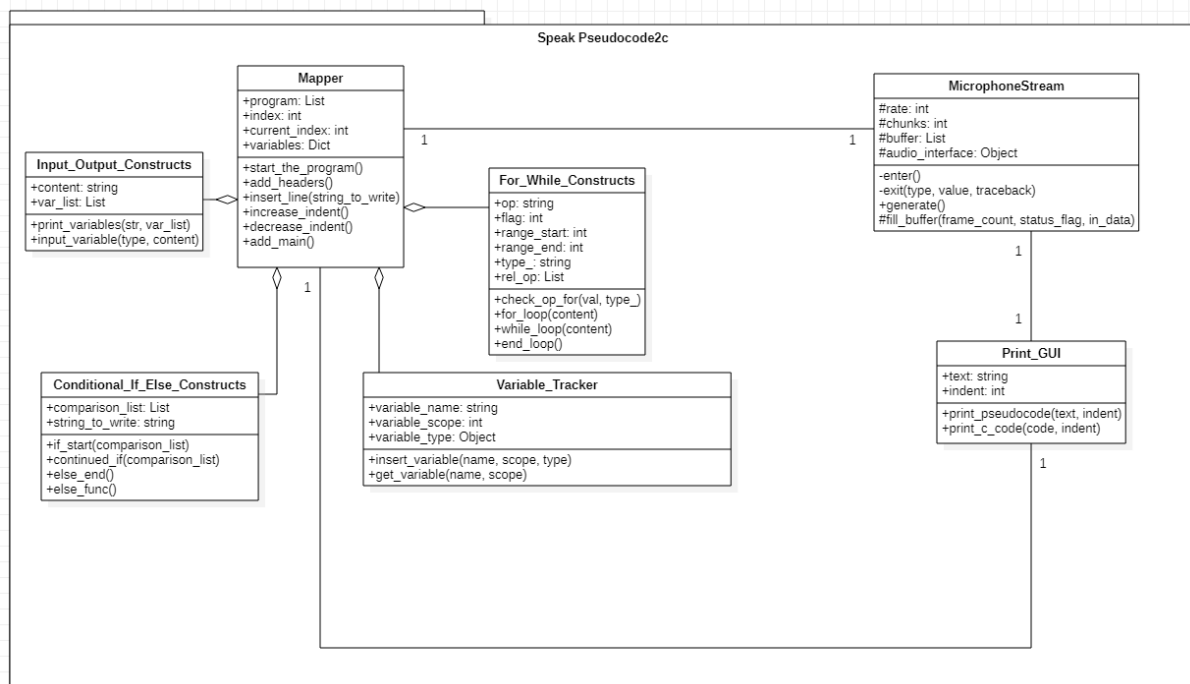


Figure 5.3(c): Master Class Diagram

5.3.1.2 Reusability Consideration

This project has three main components:

1. Speech to Text using Google Cloud API
2. Mapping Pseudocode to C code
3. Text to Speech Above three components are reused in this project. These components can also be reused in future work or integrated in other projects.

5.3.1.3 Swimlane Diagram

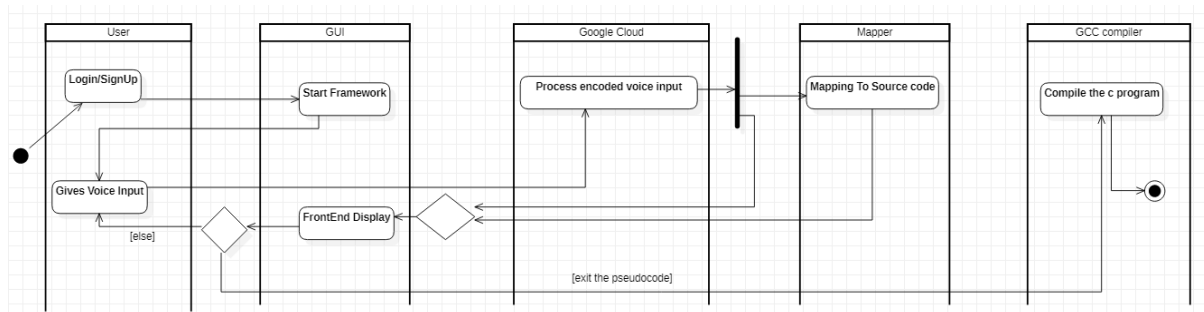


Figure 5.3(d): Activity Diagram

5.3.1.4 User Interface Diagram

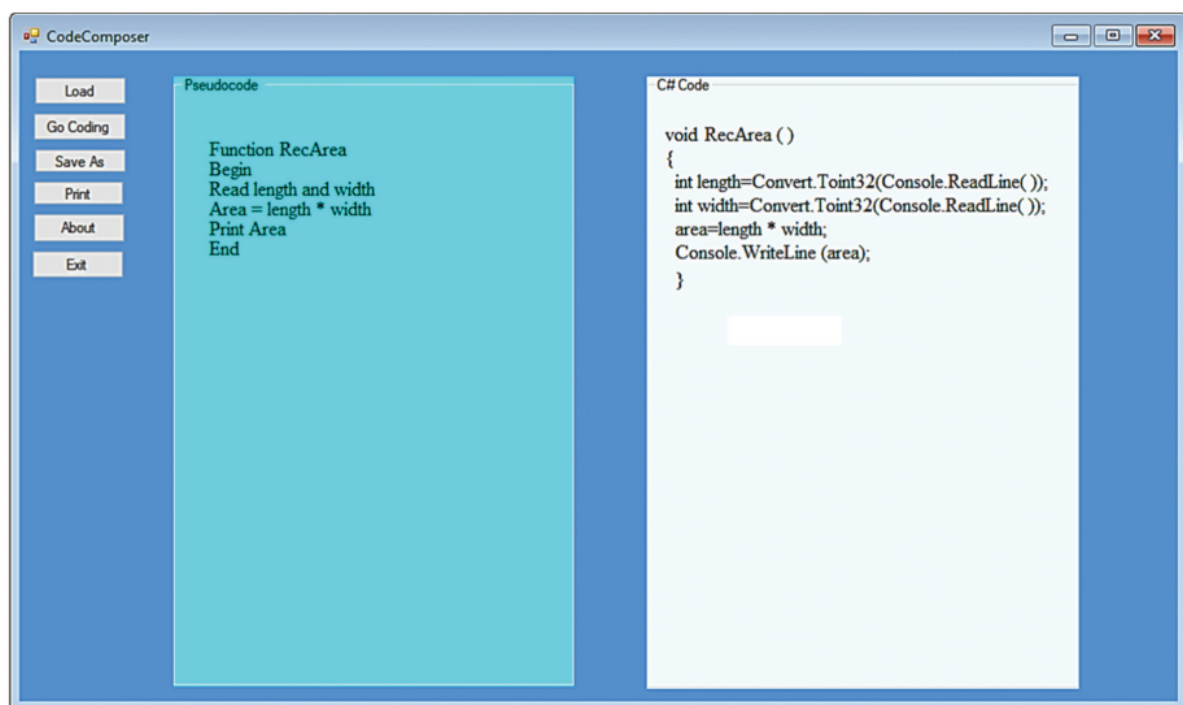


Figure 5.3(e): Activity Diagram

5.3.1.5 External Interfaces

The graphical user interface will have two text widgets in 30% to 70% ratio. The left side being 30% of the screen containing the pseudocode spoken to the program and the right side being 70% will contain the C code. These widgets will be updating their contents in real time as the

user speaks. The goal is to reduce the latency to minimum possibility and increase the accuracy of pseudocode spoken.

5.3.1.6 Packaging and Deployment Diagram

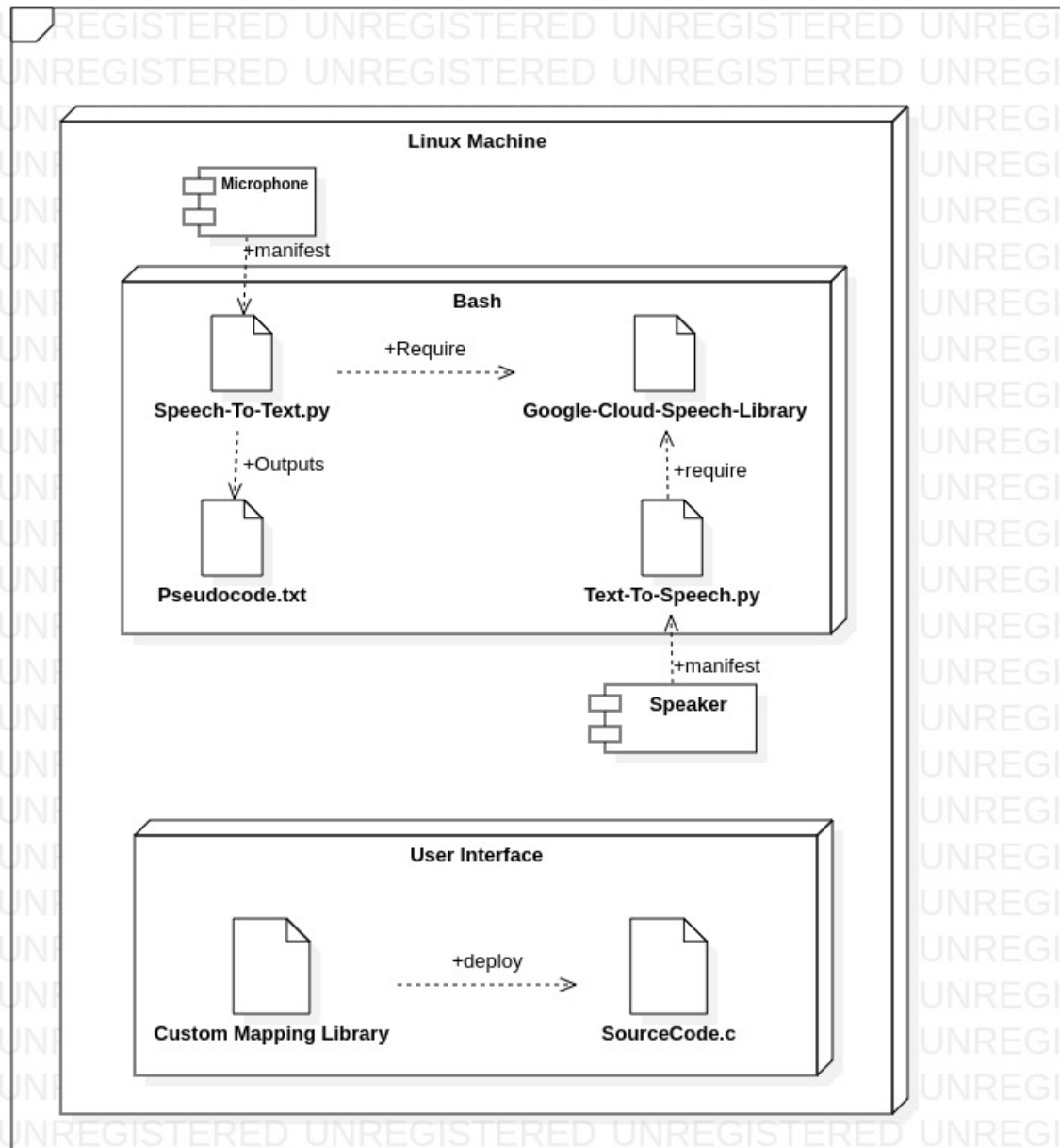


Figure 5.3(f): Deployment Diagram

5.3.2 Help

The aid planned for the system currently is the feedback provided by the system in two modes :

- Text

- Speech

We plan on providing both options for the user to configure the application and use whichever is convenient for them. The text feedback will be provided on the same pane where the user speaks and the system identifies the same. The speech feedback will just be the translation to voice for the same text. The speech option will be togglable. This help option is supposed to provide the user the correct or accurate way to use the application without any errors happening in the application. These may be feedback for the name of the function input or as simple as asking for the type of variable the user is declaring.

Also, there will be a simple documentation on basics of pseudocode which will be defined by our system but not strict by the word. The user can learn basic pseudocode language to use the application with the utmost ease and this documentation will be in the Help section of the menu bar so the user can easily spot it.

A simple user manual with all the features of the application and how to use them will be given as a tutorial in the beginning of a new registration/installation so that the user doesn't feel alien to the interface.

5.3.3 Design Details

5.3.3.1 Novelty

The system is novel as it uses user speech and directly translates it into machine translatable code which can make the process easier and faster. Current research defines many systems with the ability to convert pseudocode to machine code with a certain degree of accuracy. We intend on increasing the accuracy with the added benefit of speech so that the feature is actually usable for the user with ease.

5.3.3.2 Innovativeness

There is no commercial application which tends to do what we plan on achieving with this product. The innovativeness depends on our product mapping the pseudocode to c code with high accuracy of speech and ignoring meaningless commands.

5.3.3.3 Interoperability

Our product highly depends on the ability to use the information provided to us by the user. The speech to text engine will be highly dependent on this as well as the pseudocode to c code engine will be dependent.

5.3.3.4 Performance

The performance of the system is extremely crucial as the user will be preferring it because of the fast performance of the system and for this the speech to text engine which will be extrapolated from Google Cloud Engine will have to be extremely efficient for it to function properly.

5.3.3.5 Security

Security will play a vital role in the system as the credentials of the GCP engine should not be publicly released which can result in monetary damages to the developers. Also, the code by the user should in no way be released to the public without the permission of the user as that could result in copy and infringement of original work which we in no way intend.

5.3.3.6 Reliability

The system needs to be very reliable in terms of the converted code and the speech being accurately translated by the GCP engine. If the code converted is not interpreted in the correct way it could ruin the whole workflow of the student/developer.

5.3.3.7 Maintainability

We need to ensure maintainability of the system as by the coming day new algorithms and bugs will be addressed on the system which will have the need to be constantly updated on the product so that the user can ensure a smooth experience while coding their application.

5.3.3.8 Portability

Portability not being crucial will be inbuilt into the system as it depends on the cloud and has its own interface in the places needed ensuring the user having a similar experience on any platform where the software is installed.

5.3.3.9 Legacy to modernization

From the start of building the application we have kept a modern approach integrating cloud and UI components within the application so that the need of legacy to modernization in the future remains minimal and easy for the developers to integrate.

5.3.3.10 Reusability

Everything built into the application should be built as a component providing high cohesion and low coupling so that reusability of the components in various places is maximum.

5.3.3.11 Application compatibility

As discussed previously also, the application compatibility is crucial and built into the application as use of cloud and its own UI so that it is usable on any machine having python installed with certain modules.

5.3.3.12 Resource utilization, Etc.

The utilization maximization and efficiency is crucial to the system and we are constantly trying to optimize the workflow so that we can achieve the maximum results with minimal computation both by the user and developer end to reduce costs and efforts.

Chapter 6

LOW-LEVEL DESIGN

6.1 Introduction

6.1.1 Overview

This document focuses on the general principles of design and building the framework for Speak Pseudocode2c systems and explains the low-level concepts of different layers in the framework. This will introduce the methodology used for the evaluation and further refinement of concepts, as well as act as a guide for the implementation of the modules in reference.

6.1.2 Purpose

This document is the low-level design document for the “Speak Pseudocode2c”. The purpose of this Low-Level Design (LLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect implementation specifics and can be used as a reference manual for how the modules interact at a low level.

6.1.3 Scope

The LLD documentation presents the structure of the framework, such as the master class diagram and Design Description. The LLD uses technical terms which should be understandable to the administrators of the system. The goal is to display the code on screen via speaking. The abstract is that the users will speak the natural language pseudocode in the microphone and corresponding to that pseudocode its respective c language code will be generated. Mapping pseudocode to source code will be done automatically via the framework.

6.2 Design Constraints, Assumptions, and Dependencies

6.2.1 Constraints And Dependencies

1. Currently, we are using the Google Cloud free tier which restricts us from using their service for any commercial applications. We have to abide by their terms of service for the same.

2. Google Cloud free tier has a limitation for Speech-to-Text API which we are using in the development phase of the application, it is free for 60 minutes/month afterward it will use the credits. Compared to other cloud vendors' Speech-To-Text Google provides very little free availability of this API.
3. The hardware requirements for the system are minimal, although there is a requirement for a microphone and speaker with a working internet connection.

6.2.2 Assumptions

1. The user is familiar with the pseudo code format.
2. The user has a working internet connection.
3. The host machine has a GCC compiler installed for the execution of the c programs along with python3.
4. and google cloud python library which is necessary for running the framework.
5. There is no problem with the User Microphone and Speakers.
6. The Background noise is minimum where the client is using the application.

6.3 Design Description

We have decided to follow the function-oriented design approach where the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant tasks in the system. The system is considered as the top view of all functions.

Function-oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation. These functional modules can share information among themselves by means of information passing and using information available globally.

6.3.1 Master Class Diagram

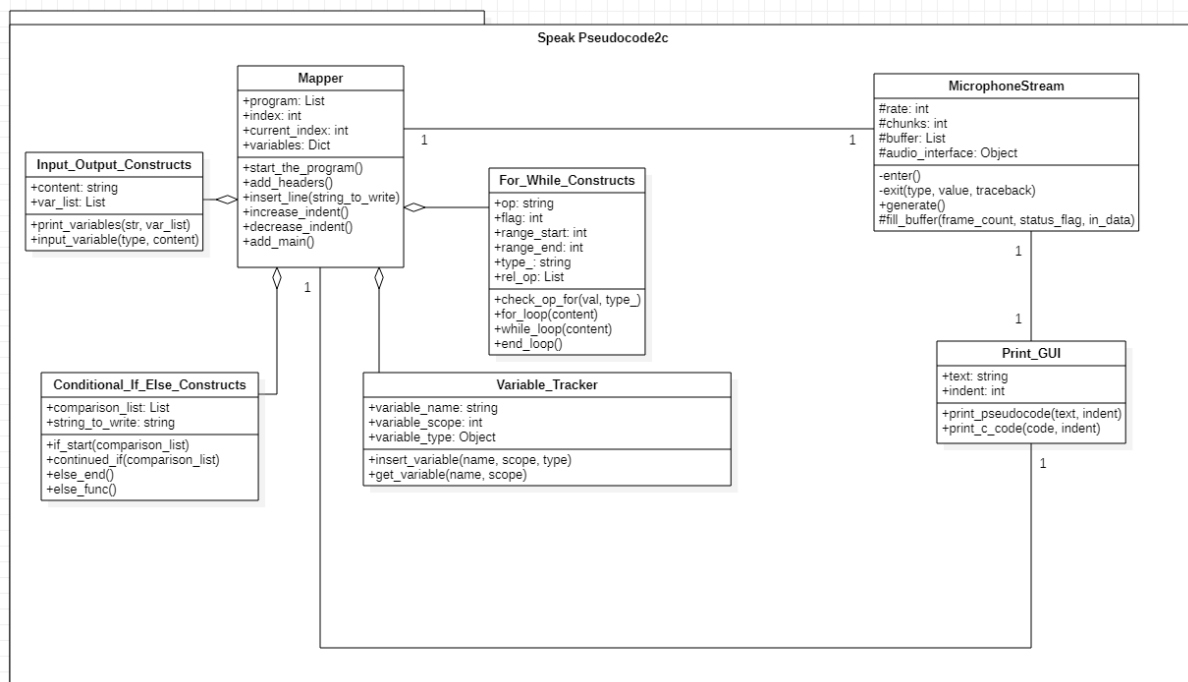


Figure 6.3(a): Master Class Diagram

6.3.2 Voice Input to Speech Module

6.3.2.1 Description

This module takes in the voice input from the user and sends it to the Google Cloud Server with valid credentials. First off, Google Cloud validates those credentials using a JSON Key and then the voice input is converted into a list of valid outputs. Then from the list of potential outputs, we chose the one which has the highest confidence according to the Google training model and then we process that output further.

6.3.3 Use Case Diagram

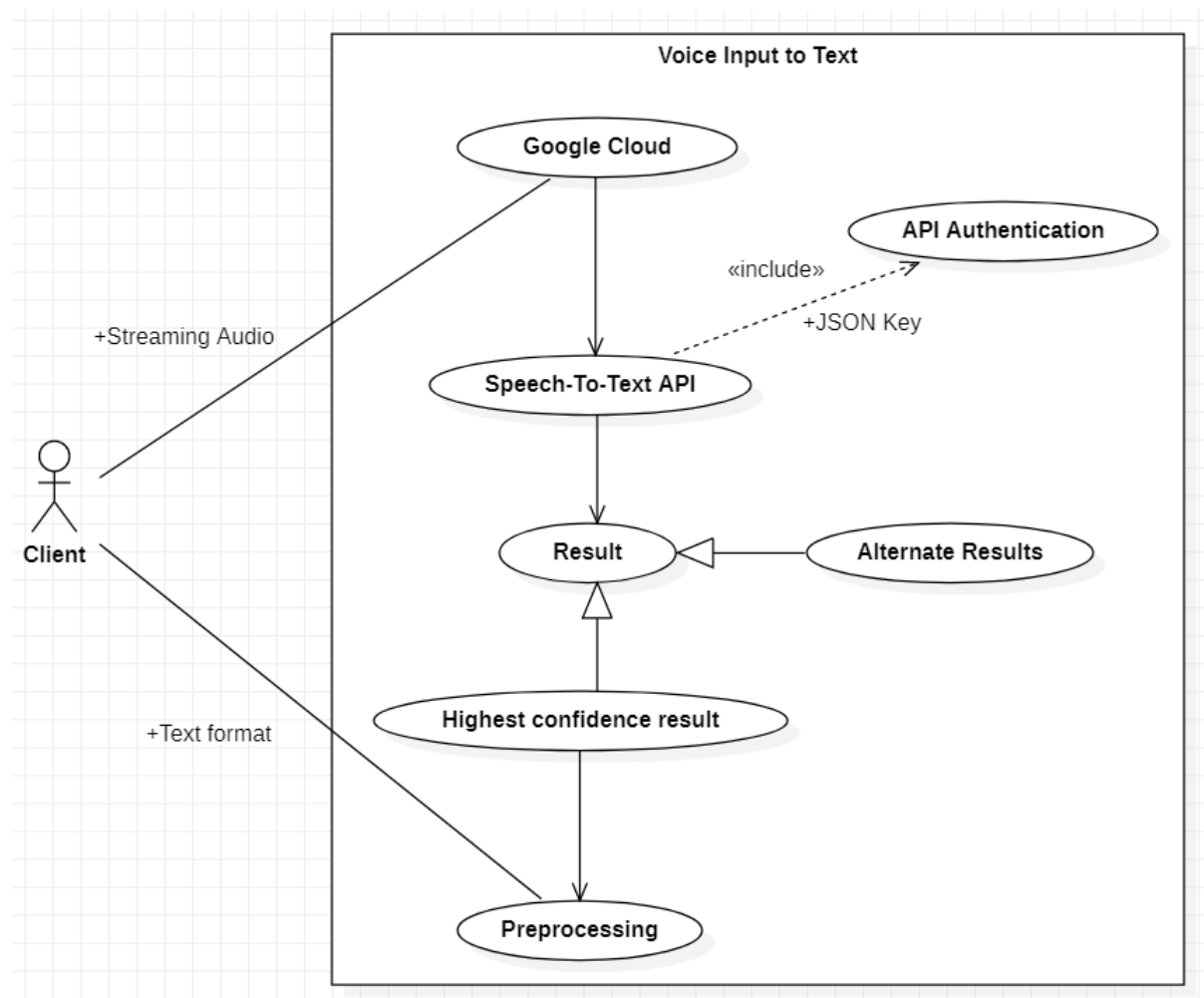


Figure 6.3(b): Use Case Diagram

6.3.4 Class Description - Mapper

6.3.4.1 Description

This class is responsible for mapping generated text to c code. It consists of methods that have mapping for constructs like input and output, for loop, while loop, and if-else.

6.3.4.2 Data members

Data Name	Data Type	Access Modifiers	Initial Value	Description
1	6	87837	787	1
2	7	78	5415	1
3	545	778	7507	1
4	545	18744	7560	1
5	88	788	6344	1

Table 6.3(b): Data Members in Mapper class

6.3.4.3 start_the_program

- **Purpose** - call functions add_headers and add_main.
- **Parameter** - mapper object

6.3.4.4 add_headers

- **Purpose** - insert the initial headers required.
- **Parameter** - mapper object
- **Input** - mapper object
- **Output** - add generated code to mapper object.

6.3.4.5 add_main

- **Purpose** - add the main function for the program.
- **Parameter** - mapper object
- **Input** - mapper object
- **Output** - add generated code to mapper object.

6.3.4.6 declare_variable

- **Purpose** - provides mapping for declaring a variable for pseudocode format - declare <variable name> <variable type> .
- **Parameter** - mapper object, generated text from speech input.

- **Input** - generated text from speech input.
- **Output** - add generated code to mapper object.

6.3.4.7 initialize_variable

- **Purpose** - provides mapping for initializing a variable for pseudocode format - initialize
<variable name> = <variable value> .
- **Parameter** - mapper object, generated text from speech input.
- **Input** - generated text from speech input.
- **Output** - add generated code to mapper object.
- **Exception** - ValueError

6.3.4.8 input_variable

- **Purpose** - provides mapping to input a variable for pseudocode formats -
 1. input <variable name> <variable type>
 2. input <variable names> <variable types>
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.

6.3.4.9 assign_variable

- **Purpose** - provides mapping for assigning a variable for pseudocode format - <variable
result> = <variable 1> <operator> <variable 2>
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.
- **Exception** - VariableNotDeclared

6.3.4.10 print_variables

- **Purpose** - provides mapping for printing a string or a variable . It handles the following formats -
 1. print variable <variable name>
 2. print <string>
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.

6.3.4.11 continued_if

- **Purpose** - provides mapping for normal and nested if-else statements . It handles the following formats -
 1. if <variable1> <operator> <variable2>
 2. else if <variable1> <operator> <variable2>
 3. else
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.

6.3.4.12 while_loop

- **Purpose** - provides mapping for while statements . It handles the following formats -
 1. while <variable>
 2. while <variable> <operator> <variable>
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.

- **Exceptions** - VariableNotDeclared

6.3.4.13 for_loop

- **Purpose** - provides mapping of all for statements . It handles the following formats -
 1. for iterator [anything] (optional start_point) till end_point(char or int) (optional increment/decrement by int).
 2. for iterator in range from alphanumeric till alphanumeric increment by integer.
 3. for iterator in range alphanumeric till alphanumeric.
 4. for iterator in range till alphanumeric.
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.
- **Exceptions** - VariableNotDeclared

6.3.4.14 end_func

- **Purpose** - To handle ending of constructs like while, for and if.
- **Parameter** - mapper object
- **Input** - mapper object
- **Output** - add closing braces to mapper object.

6.3.4.15 get_program_list

- **Purpose** - Return the contents of the program in mapper object.
- **Parameter** - mapper object
- **Input** - mapper object
- **Output** - return program

6.3.4.16 comment

- **Purpose** - Enable user narration by commenting lines which are not intended to be part of the code.
- **Parameter** - mapper object, list of generated text from speech input.
- **Input** - list of generated text from speech input.
- **Output** - add generated code to mapper object.

6.3.4.17 break_stmt

- **Purpose** - Insert break statement wherever required.
- **Parameter** - mapper object.
- **Input** - mapper object.
- **Output** - add break statement to mapper object.

6.3.4.18 continue_stmt

- **Purpose** - Insert continue statement wherever required.
- **Parameter** - mapper object.
- **Input** - mapper object.
- **Output** - add continue statement to mapper object.

6.3.4.19 process_input

- **Purpose** - process the speech input and send it to the appropriate function for further conversion.
- **Parameter** - mapper object, speech input in string format.
- **Input** - mapper object, speech input in string format.
- **Output** - return program from mapper object.

6.3.5 Class Description - Pseudocode2c

6.3.5.1 Description

This class is used to implement the graphical user interface (gui) for the framework. The class contains two vertical split text boxes which run parallelly with the help of threads. The left text box is used to interact with Google Speech to text, it takes speech input customized pseudocode and converts it into text. The right text box displays the c language source code. It takes the pseudocode in text format and passes it to the Mapper class.

6.3.5.2 Data members

Data Name	Data Type	Access Modifiers	Initial Value	Description
1	6	87837	787	1
2	7	78	5415	1
3	545	778	7507	1
4	545	18744	7560	1
5	88	788	6344	1

Table 6.3(c): Data members in Pseudocode2c class

6.3.5.3 callback

- **Purpose** - This function contains the piece of code which will run after the thread terminates, usually, garbage cleaning.
- **Input** - class object.
- **Output** - Closes the Tkinter.

6.3.5.4 run

- **Purpose** - Thread starts running from this point. The code written under this will be executed first.
- **Input** - class object
- **Output** - Build various widgets (text box, frame, and button) of the framework .

6.3.5.5 save_code

- **Purpose** - When the save button is clicked then it executes the code under this function. It stores the text content in the right text box and writes it into the .c file.
- **Input** - class object.
- **Output** - Output - .c source program file.

6.3.5.6 compile_program

- **Purpose** - When the compile button is clicked then it executes the code under present in the right text box, but first the text needs to be stored in the .c file.
- **Input** - class object.
- **Output** - Run .c program and display result on the terminal.

6.3.5.7 remove_junk

- **Purpose** - When the undo button is clicked then it deletes the last number of lines written in the right text box i.e the conversion of pseudocode to source code for a particular line.
- **Input** - class object and count of lines written.
- **Output** - deletes the previously written lines.

6.3.5.8 exit_code

- **Purpose** - When the exit button is clicked then it executes the code under this function. It destroys all the widget created by the run function.
- **Input** - class object.
- **Output** - destroy all the widgets.

6.3.5.9 insert_lhs

- **Purpose** - It listens to the output of Google speech-To-Text and writes the text format pseudocode in the text box.
- **Input** - class object and text to be written in the left text box.

- **Output** - writes the pseudocode in left text box.

6.3.5.10 insert_rhs

- **Purpose** - It passes the output of Google speech-To-Text to the Mapper class which returns the c language source code and writes the source code in the right text box.
- **Input** - class object and text to be written in the right text box.
- **Output** - writes the c source code in right text box.

6.3.5.11 show_alert

- **Purpose** - On occurrence of any error or exception, it prompts an alert box showing the type of exception occurred and alerts users about the mistake committed.
- **Input** - class object and text to be written in the alert box.
- **Output** - Prompt the alert box if any error or exception occurs in the c program.

6.3.6 Class Description - MicrophoneStream

6.3.6.1 Description

This class opens a recording stream as a generator yielding the audio chunks. It receives the audio data, encodes it and sends it to Google cloud for processing. After processing, it receives the text output from Google Speech-To-Text API and outputs it to the graphical user input (gui).

6.3.6.2 Data members

Data Name	Data Type	Access Modifiers	Initial Value	Description
1	6	87837	787	1
2	7	78	5415	1
3	545	778	7507	1
4	545	18744	7560	1
5	88	788	6344	1

Table 6.3(d): Table to show data members in Mapper class.

6.3.6.3 `__enter__`

- **Purpose** - It creates a thread-safe buffer of audio data and runs the audio stream asynchronously to fill the buffer object. This is necessary so that the input device's buffer doesn't overflow while the calling thread makes network requests, etc.
- **Input** - class object
- **Output** - class object and start listening to the audio and also, starts writing data to the buffer.

6.3.6.4 `__exit__`

- **Purpose** - Signal the generator to terminate so that the client's `streaming_recognize` method will not block the process termination.
- **Input** - class object, value and traceback
- **Output** - Closes the audio listener and clear the buffer.

6.3.6.5 `_fill_buffer`

- **Purpose** - Continuously collect data from the audio stream, into the buffer.
- **Input** - class object, `in_data`, `frame_count`, `time_info`, and `status_flag`.
- **Output** - Writes data to the buffer.

6.3.6.6 `generator`

- **Purpose** - Use a blocking `get()` to ensure there's at least one chunk of data, and stop iteration if the chunk is `None`, indicating the end of the audio stream.
- **Input** - class object
- **Output** - Yields the data in binary raw format.

6.3.6.7 `listen_print_loop`

- **Purpose** - Iterates through server responses and prints them. The response passed is a generator that will block until a response is provided by the server. Each response may

contain multiple results, and each result may contain multiple alternatives. Here we print only the transcription for the top alternative of the top result.

- **Input** - responses from Google Speech-To-Text
- **Output** - Prints the customized pseudocode on the gui left text box.

6.4 Sequence Diagram

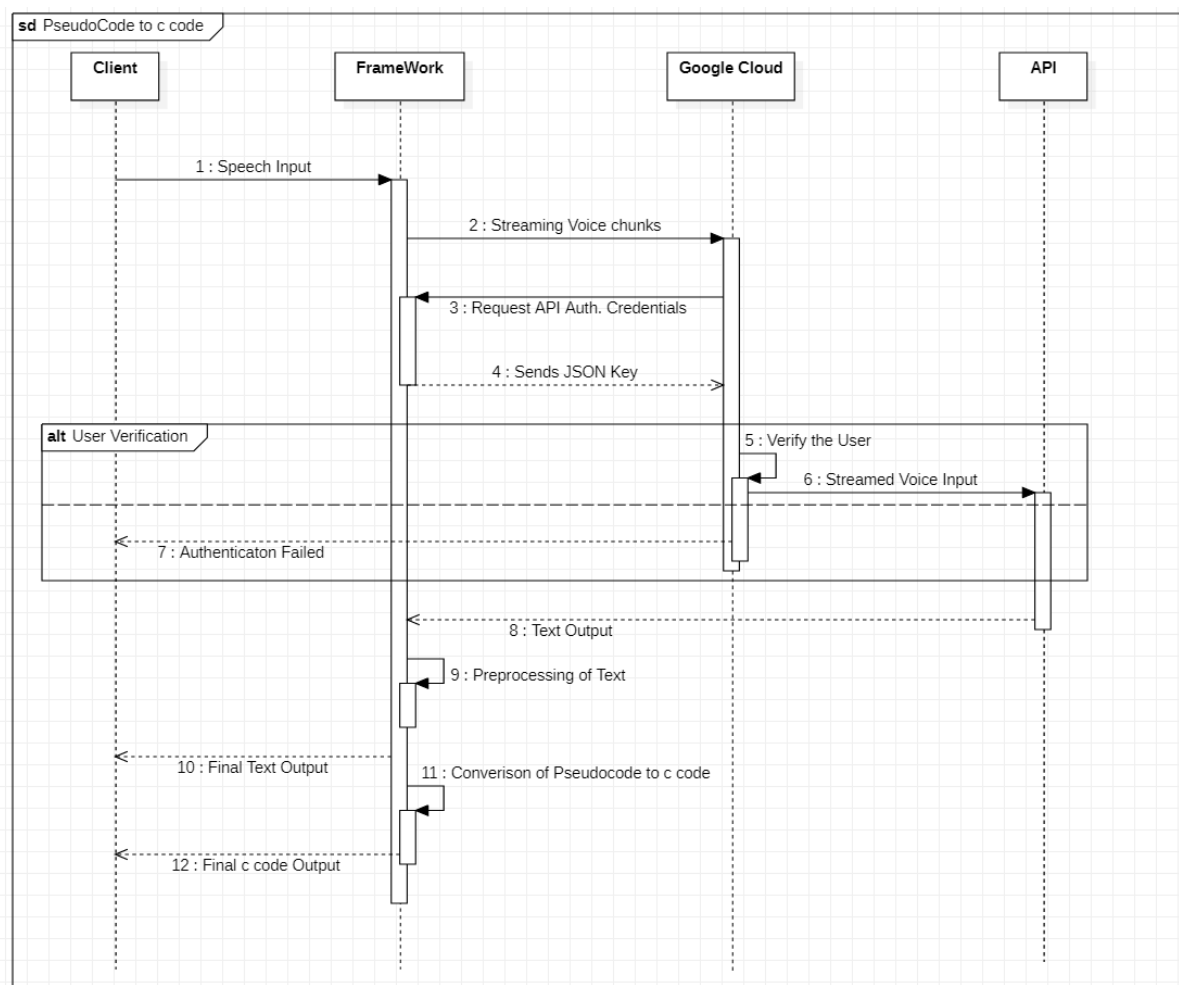


Figure 6.4(c): Sequence Diagram

6.5 Packaging and Deployment Diagrams

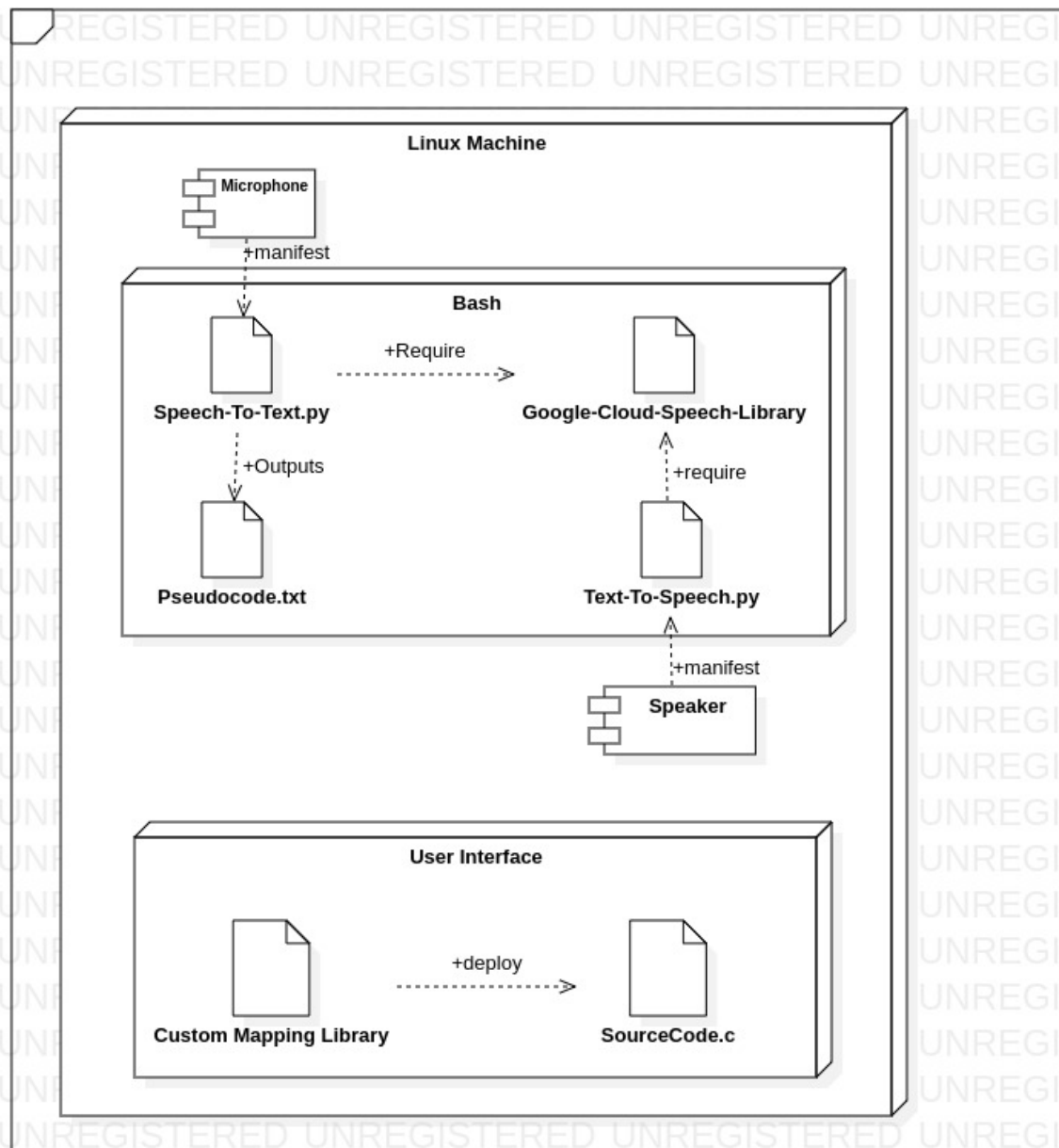


Figure 6.5(d): Deployment Diagram

Chapter 7

Proposed Methodology

[This section clearly defines the constraints involved in the design with reasons. If these constraints can be overcome by certain assumptions, they will be stated too. Dependencies, if any, in the design will be mentioned clearly.]

7.1 Algorithm and Pseudocode

[Add details on the Algorithm used and write Pseudocode to explain the logical workflow of the project]

7.2 Implementation and Results

[Add details of your approach, experimental results. Details of how the initial approaches were fine tuned and their results Discuss the results and the progress so far.]

7.3 Further Exploration Plans and Timelines (optional)

[Add information on changes, if any, in your research approach. Timelines for changing the approach.]

References

- [1] P. Sirikongtham and W. Paireekreng, “Improving speech recognition using dynamic multi-pipeline api,” in *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*. IEEE, 2017, pp. 1–6.
- [2] T. Dirgahayu, S. N. Huda, Z. Zuhri, and C. I. Ratnasari, “Automatic translation from pseudocode to source code: A conceptual-metamodel approach,” in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*. IEEE, 2017, pp. 122–128.
- [3] A. Abujabal and J. Gaspers, “Neural named entity recognition from subword units,” *arXiv preprint arXiv:1808.07364*, 2018.
- [4] A. T. Imam and A. J. Alnsour, “The use of natural language processing approach for converting pseudo code to c# code,” *Journal of Intelligent Systems*, vol. 29, no. 1, pp. 1388–1407, 2020.
- [5] L. Haowen, L. Wei, and L. Yin, “An xml-based pseudo-code online editing and conversion system,” in *IEEE Conference Anthology*. IEEE, 2013, pp. 1–5.