INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

QuorumGen: Voice Assisted Source Code Generating tool for Students with Upper Limb Disability

A dissertation by

Suvetha Suvendran

Supervised by

Mr. R. Sivaraman

Submitted in partial fulfilment of the requirements for the BEng (Hons) in Software Engineering degree at the University of Westminster.

May 2023

Declaration

I hereby declare that this dissertation and all of its related sections are an outcome of my own research and that none of them have ever been or are now being submitted to another university or organization as part of any degree or other certification program. Facts that were taken from reputable outside sources have been properly cited.

Student Full Name: Suvetha Suvendran

Registration Number: w1790157 | 2019684

Signature: S.Suvetha **Date: 10th May 2023**

Abstract

Students with disabilities require specialized support and accessible technologies in order to participate and succeed in their academics without facing attitude obstacles. Quorum is a high-level, object-oriented programming language designed to be accessible to learn programming concepts for differently abled students. Upper limb disabled students have inaccessibility due to the unavailability of a tool to understand natural language voice commands to generate Quorum code blocks since they cannot code by the conventional method which relies on the dexterity of human hands.

Students with upper limb disabilities find it inspiring to be able to write code using their voice because it will give them more confidence to pursue their education in software development field. The chosen problem will be resolved by developing a tool that understands natural language voice instructions to generate Quorum language code blocks to provide accessibility for them to code by voice and easily learn the programming concepts using natural language processing and deep learning that aims to convert spoken commands or instructions into executable source code with custom dataset for this specific task of generating Quorum code snippets.

The core implementation of this project is the natural language command to Quorum source code generation with custom dataset. The model performance accuracy was 81.62% for code generation with the bounded scope of syntax and structure of Quorum language syntax trained, tested, and evaluated. Holdout validation was used in this research to evaluate the selected model

Keywords: Voice recognition, Natural Language Processing (NLP), Code generation, Upper limb disabled student, Accessibility, Deep learning, Programming education

Subject Descriptors:

- Computing methodologies $\longrightarrow$ Artificial Intelligence $\longrightarrow$ NLP $\longrightarrow$ Speech Recognition
- Computing methodologies $\longrightarrow$ Machine learning $\longrightarrow$ Machine learning approaches $\longrightarrow$ Neural networks

Acknowledgement

It gives me great pleasure to thank my boss, Mr. Sivaraman Raghu, from the bottom of my heart. His constant encouragement, motivation, and direction allowed me to successfully complete the research. I want to thank Mr. Guhanathan Poravi for his insightful advice and inspiration. I express my profound gratitude to the Informatics Institute of Technology's administration, my instructors, and my coworkers. Finally, I want to express my gratitude to my friends and family for always supporting and believing in me.

Table of Contents

Abstract ii
List of Tables x
List of Figures xi
List of Abbreviations xii
CHAPTER 1: INTRODUCTION 1
1.1 Chapter Overview 1
1.2 Problem Domain 1
1.2.1. Education assistance for the differently abled students 1
1.2.2. Challenges in learning programming 1
1.2.3. Programming by Voice 2
1.3. Problem Definition 2
1.3.1. Problem Statement 3
1.4. Research Motivation 3
1.5. Research Gap 3
1.6. Contribution to the Body of Knowledge 4
1.6.1. Contribution to the Research Domain 4
1.6.2. Contribution to the Problem Domain 4
1.7. Research Challenge 5
1.8. Research Questions 6
1.9. Research Aim 6
1.10. Research Objectives 6
1.11. Project Scope 9
1.11.1. In-scope 9
1.11.2. Out-scope 10
1.11.3. Prototype Feature Diagram 10
1.12. Chapter Summary 10
CHAPTER 2: LITERATURE REVIEW 11
2.1. Chapter Overview 11
2.2. Concept Map 11

2.3. Problem Domain 11
2.3.1. Upper Limb Disability 12
2.3.2. Programming and Upper Limb Disabled Students 12
2.3.3. Quorum Programming Language 13
2.3.4. Code generation 14
2.3.5. Natural Language Processing (NLP) 14
2.3.6. Natural Language Commands to Code Generation 15
2.4. Existing Work 15
2.4.1. Generative AI and code generation 15
2.4.2. Rule-based systems for code generation 16
2.4.3. Machine learning-based systems for code generation 16
2.4.4. Hybrid systems for code generation 17
2.4.5. Domain-specific approaches for code generation 17
2.4.6. Augmented Intelligence for code generation 18
2.4.7. Transfer Learning for code generation 18
2.4.8. Combination of Transfer learning and Machine learning-based systems 19
2.4.9. Summary of Existing work 20
2.5. Technological Review 22
2.5.1. Transformer based language models 22
2.5.2. T5 (Text-to-Text Transfer Transformer) 25
2.6. Evaluation and benchmarking 26
2.6.1. Holdout Validation 28
2.7. Chapter Summary 28
CHAPTER 3: METHODOLOGIES 29
3.1. Chapter Overview 29
3.2. Research Methodology 29
3.3. Development Methodology 30
3.4. Project Management Methodology 30
3.4.1. Schedule 30
3.5. Resources 31

3.5.1. Hardware Resources 32
3.5.2. Software Resources 32
3.5.3. Technical Skills 32
3.5.4. Data Requirements 32
3.6. Risks and Mitigation 32
3.7. Chapter Summary 33
CHAPTER 4: SOFTWARE REQUIREMENTS SPECIFICATION 34
4.1. Chapter Overview 34
4.2. Rich Picture Diagram 34
4.3. Stakeholder Analysis 35
4.3.1. Stakeholder onion model 35
4.3.2. Stake holder Viewpoints 35
4.4. Selection of Requirement Elicitation 37
4.5. Discussion of Findings 38
4.5.1. Literature Review 38
4.5.2. Interviews 39
4.5.3. Observation 39
4.5.4. Brainstorming 40
4.5.5. Summary of Findings 40
4.6. Context Diagram 42
4.7. Use case diagram 42
4.8. Use case descriptions 42
4.9. Requirements 43
4.9.1. Functional Requirements 44
4.9.2. Non-Functional Requirements 45
4.10. Chapter Summary 45
CHAPTER 5: SOCIAL, LEGAL, ETHICAL AND PROFESSIONAL ISSUES (SLEP) 46
5.1. Chapter Overview 46
5.2. SLEP issues and mitigation 46
5.2.1. Social Issues 46

5.2.2. Legal Issues ... 46
5.2.3. Ethical Issues ... 46
5.2.4. Professional Issues ... 47
5.3. Chapter Summary ... 47
CHAPTER 6: DESIGN ... 48
6.1. Chapter overview ... 48
6.2. Design Goals ... 48
6.3. High level Design ... 49
6.3.1. Architecture Diagram ... 49
6.3.2. Discussion of layers of the Architecture ... 50
6.4. Low-level Design/ System Design ... 51
3.4.1 Choice of design paradigm ... 51
6.5. Design Diagrams ... 51
6.5.1. Component Diagram ... 51
6.5.2. Data Flow Diagram ... 52
6.5.3. Algorithmic Analysis ... 52
6.5.4. User Interface Design ... 54
6.5.4.1. Low level fidelity wireframe diagram ... 54
6.5.4.2. High level fidelity diagram ... 54
6.5.5. System Process Flow Chart ... 55
6.6. Chapter Summary ... 55
CHAPTER 7: IMPLEMENTATION ... 56
7.1. Chapter Overview ... 56
7.2. Technology Selection ... 56
7.1.1. Technology Stack ... 56
7.1.2. Data-set Selection ... 56
7.2.3. Development Frameworks ... 58
7.2.4. Programming Languages ... 58
7.2.5. Libraries ... 59
7.2.6. IDE ... 59

7.2.7. Summary of Technology Selection 59
7.3. Implementation of the Core Functionality 60
7.4. User Interface 63
7.5. Chapter Summary 63
CHAPTER 8: TESTING 64
8.1. Chapter Overview 64
8.2. Objectives and Goals of Testing 64
8.3. Testing Criteria 65
8.4. Model Testing & Evaluation 65
8.5. Holdout Validation 67
8.6. Functional Testing 67
8.7. Module and Integration Testing 68
8.8. Non-Functional Testing 69
8.9. Limitations of the testing process 69
8.10. Chapter Summary 69
CHAPTER 9: EVALUATION 70
9.1. Chapter Overview 70
9.2. Evaluation Methodology and Approach 70
9.3. Evaluation Criteria 70
9.4. Self-Evaluation 71
9.5. Selection of the Evaluators 72
9.5.1. End Users 72
9.5.2. Experts 72
9.6. Evaluation Result 73
9.7. Limitations of Evaluation 74
9.8. Evaluation on Functional Requirements 74
9.9. Evaluation on Non-Functional Requirements 75
9.10. Chapter Summary 75
CHAPTER 10: CONCLUSION 77
10.1. Chapter Overview 77

10.2. Achievements of Research Aims & Objectives 77
10.2.1. Research Aims 77
10.2.2. Research Objectives 77
10.3. Utilization of Knowledge from the Course 78
10.4. Use of Existing Skills 78
10.5. Use of New Skills 78
10.6. Achievement of Learning Outcomes 79
10.7. Problems and Challenges Faced 79
10.8. Deviations 80
10.8.1. Scope related Deviations 80
10.8.2. Schedule related deviations 80
10.9. Limitations of the Research 81
10.10. Future Enhancements 81
10.11. Achievement of the contribution to body of knowledge 81
1.12.1. Contribution to the Research Domain 81
1.12.2. Contribution to the Problem Domain 82
10.12. Concluding Remarks 82
REFERENCES I
Appendix A – Gantt Chart V
Appendix B - SRS VI
Appendix C – Design VIII
Appendix C1 – Algorithmic Analysis VIII
Appendix C2 – Algorithmic Analysis IX
Appendix D – Testing X
Appendix D1 - T5-mini Evaluation X
Appendix D2 - T5-mini Evaluation XI
Appendix E - Evaluation XII

List of Tables

Table 1: Research Challenge 5
Table 2:Research Objectives........ 6
Table 3: Related Work........ 20
Table 4:Langauge model review........ 23
Table 5:Research Methodology 29
Table 6:Deliverables and dates 31
Table 7:Risk and mitigation........ 33
Table 8: Risk and Mitigation 33
Table 9: Stakeholder Viewpoints........ 35
Table 10: Selection of Requirement Elicitation........ 37
Table 11: Literature Review Findings 38
Table 12: Interview Findings........ 39
Table 13: Observation Findings........ 39
Table 14: Brainstorming Findings 40
Table 15: Summary of Findings 41
Table 16: Use Case 01 42
Table 17: Use Case 02 43
Table 18: MoSCoW Table........ 43
Table 19:Functional Requirements........ 44
Table 20: Non Functional Requirements........ 45
Table 21: Design Goals........ 48
Table 22: Discussion of layers of the Architecture........ 50
Table 23: Development Frameworks........ 58
Table 24: Libraries selection and Justification 59
Table 25: IDE Selection and Justification 59
Table 26: Summary of Technology Selection 59
Table 27:Holdout validation scores 67
Table 28:Model and Integrated Testing........ 68
Table 29:Evaluation Criteria........ 70
Table 30:Self Evaluation........ 71
Table 31:End Users........ 72
Table 32:Experts 72
Table 33Evaluation Result........ 73
Table 34:Evaluation of functional Requirements 74
Table 35:Evaluation of non-functional Requirements........ 75
Table 36:Achievements of research objectives........ 77
Table 37:Achievemnets of learning outcome 79
Table 38:Problems and challenges faced........ 79
Table 39:Requirment gathering from stakeholders........ VI

List of Figures

Figure 1:Prototype Feature Diagram 10
Figure 2: Concept Map 11
Figure 3: Quorum Code snippet........ 13
Figure 4: Rich Picture Diagram 34
Figure 5: Stakeholder Onion Model 35
Figure 6: Context Diagram 42
Figure 7: Use Case Diagram-........ 42
Figure 8: Architecture Diagram 49
Figure 9: Component Diagram 51
Figure 10: Level 1 - DFD........ 52
Figure 11: Level 2 - DFD........ 52
Figure 12: Algorithm analysis 53
Figure 13: Perplexity Equation 54
Figure 14: BLEU Equation 54
Figure 15: Activity Diagarm 55
Figure 16: Technology Stack 56
Figure 17: Dataset snippet 57
Figure 18: Model Training........ 60
Figure 19: Defining dataset........ 61
Figure 20: Create data loaders 61
Figure 21: Add new token........ 62
Figure 22: Trianing T5-small with custom dataset 62
Figure 23: Testing with input command........ 63
Figure 24: User Interface 63
Figure 25: Evaluation........ 65
Figure 26: Classification report code........ 66
Figure 27: Classification report 66
Figure 28: Classification report 67
Figure 29: Functional Testing 1........ 67
Figure 30:Functional Testing 2 68
Figure 31: Functional Testing 3........ 68
Figure 32:Gantt chart V
Figure 33: Algorithm analysis appendix........ VIII
Figure 34: T5 variants hyperparameters VIII
Figure 35:Low level fidelity wireframe diagram........ IX
Figure 36:High level fidelity wireframe diagram IX
Figure 37:T5-mini evaluation X
Figure 38:T5-base Evaluation........ XI
Figure 39:Tool Evaluation 1 XII

Figure 40:Tool Evaluation 2 XII
Figure 41: Tool Evaluation 3 XII
Figure 42:Tool Evaluation 4 XIII
Figure 43:Tool Evaluation 5 XIII
Figure 44:Tool Evaluation 6 XIII

List of Abbreviations

Acronym	**Description**
IDE	Integrated development environment
AI	Artificial Intelligence
ULD	Upper Limb Disability
NLP	Natural Language Processing
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network

CHAPTER 1: INTRODUCTION

1.1 Chapter Overview

This chapter provides an overview of the research carried out to develop a voice assisted natural language commands to source code generating tool for students with upper limb disability. It further outlines the description of the problem domain, research aims, objectives, novelty of the problem and solution, identified research gap, and the contribution to the problem and research domains are described.

1.2 Problem Domain

1.2.1. Education assistance for the differently abled students

Students with disabilities who have limited or completely no use of their arms, hands, or fingers are considered upper limb disabled. Several disorders, including cerebral palsy, spinal cord injuries, illnesses like multiple sclerosis, birth deformities, stroke, muscular dystrophy, can result in these impairments (Cody L. McDonald, 2019). The capacity to carry out daily duties and activities, including education, will be impacted by upper limb limitations. Education assistance for the differently abled students is an important aspect of elevating inclusion, efficiency of the accessibility, and equity in education (Miyazaki, 2019). Both the population and technological advances in the programming industry are accelerating but considering the differently abled students they need assistive technologies and accommodations for their academic performance without attitudinal barriers. This constraint prevents this population from getting into the programming industry (F. H. Boot, 2018).

1.2.2. Challenges in learning programming

In recent years, programming has undergone tremendous changes but students with upper limb disabilities face several challenges when trying to learn programming (Lagergren, 2021). Inadequate assistance of technologies and tools to learn programming, efficiency of the accessibility and lacking inclusivity are major barriers to be concerned. As a physical act of typing, it is challenging for students with upper limb disabilities to access these programming tools with conventional usage of mouse and keyboard (Ludi, 2022). The majority of computer programs and environments are made to communicate with users using hand and finger gestures. This kind of environment which relies on the dexterity of human hands is challenging to learn programming for students with upper limb disabilities (Cody L. McDonald, 2019). The highly integrated tools for programming cannot be accessed by the conventional way of using a mouse and keyboard for the students with above mentioned disability (F. H. Boot, 2018).

1.2.3. Programming by Voice

When considering the voice-based systems for upper limb disabled students it is vital to provide access by voice to all features and tasks performed by the conventional method as other programming students (Miyazaki, 2019). Learning programming and syntax consumes more time and voice strains for students. There are higher possibilities to make mistakes and for wrong representation of words while converting from voice to text (Nizzad, 2022). Text entering, navigation, selecting texts, replacement, refactoring, cancellation, and bug fixing cannot be done by the upper-limb-disabled students by accessing it using keyboard and mouse (Cody L. McDonald, 2019). Windows Speech Recognition and Dragon NaturallySpeaking are two examples of popular voice recognition software to convert voice inputs to text but dictating each and every syntax and words of a whole program is frustrating and exhausting. Also, students who start learning programming concepts will find it easier to understand syntax and concepts related to the natural language commands. Since programming languages typically use indents, syntax notations, and strict syntax restrictions, converting natural language commands to code blocks for simple syntax programming languages will make learning and understanding programming efficient and accessible for students with upper limb disabilities (Joe Zhang, 2022).

1.3. Problem Definition

The benefits of having access to computer programming and learning programming concepts should be available to everyone equally. The differently abled students need special assistance and accessible tools to involve and perform in their academics without attitudinal barriers (Wagner, 2014). Instead of adjusting to the unique needs and preferences of differently abled students, the majority of systems continue to be built on one-size-fits-all methodologies. Physical barriers, efficiency of accessibility, lack of adaptive technology, and inadequate training and support make it difficult for differently abled students to learn programming (Lagergren, 2021). It is difficult to dictate every word, quotation mark, parenthesis, indent, and semicolon in the program code in order to convert it from natural voice commands to source code and understand the programming language concepts for students in the beginning level(S. Hossain, 2021). Differently abled students who can only learn programming by voice find it particularly challenging and straining due to this issue. Quorum is a high-level, object-oriented programming language created to make programming concepts understandable for differently abled students but currently no tool available that can directly convert natural language voice commands into Quorum code blocks (Koulouri, 2014). The lack of a corpus of general language voice commands to generate code blocks and difficulty in understanding the programming concepts prevents differently abled students from learning programming in an effective and accessible way (Miyazaki, 2019).

1.3.1. Problem Statement

Barrier and complexity in accessing the programming learning tools and understanding the programming concepts as students with upper limb disabilities.

1.4. Research Motivation

Everyone should have equal access to the advantages of accessing computer programming and learning programming concepts and languages. Students with upper limb limitations find it difficult to use these programming tools with traditional mouse and keyboard usage because typing is a physical act. There aren't tools to generate source code for the Quorum programming language created to make programming concepts understandable for differently abled students for natural voice commands. The importance to provide education assistance for the differently abled students and elevating inclusion, efficiency of the accessibility, and equity in education of learning programming was the key aspect of developing a tool to generate source code for natural voice commands.

1.5. Research Gap

Accessible programming learning tool for students with motor disability and upper limb disability has considerable research gaps. Source code generation with voice assisted programming development environment using a machine learning approach is available where every word, quotation mark, parenthesis, indent, and semicolon in the program code should be dictated (Nizzad, 2022). A natural language commands to generate source code for simple syntax programming languages like Quorum language is lacking to make learning programming effective and accessible for students with upper limb disabilities (Ayushi Trivedi, 2018). According to the researches so far, there isn't a voice assisted source code generating tool for Quorum language which is an evidence based language created for differently abled students to learn programming (M. G. Prakash, 2022). This tool is not available in the existing voice assisted programming learning tools and particularly for the quorum language (Miyazaki, 2019). Existing code generating tools are mainly trained to generate widely used programming languages. None of the existing natural language understanding tools which generates programming code can generate quorum programming language code. A voice recognition tool which converts natural language commands to Quorum code blocks will provide an efficient and accessible learning environment for students with upper limb disabilities (Apostolos, 2017).

1.6. Contribution to the Body of Knowledge

1.6.1. Contribution to the Research Domain

Code generation is an active area of research within the field of programming languages and has applications in a wide range of domains, from software engineering to robotics. This project involves using NLP techniques and transformer-based neural network architecture to convert natural language voice commands into executable Quorum source code. Custom dataset to train the T5-small variant of the transformer architecture, which is a type of neural network designed for natural language processing tasks and the trained model to generate Quorum programming language source code for natural language commands are the contribution to the research domain.

1.6.2. Contribution to the Problem Domain

It is difficult for individuals with upper limb limitations, and it is impossible in the majority of the cases to access keyboards by hand and perform coding. A voice to source code generating tool can greatly increase accessibility and inclusivity by allowing them to code using their voice. Individuals with physical impairments or constraints that make typing challenging or impossible can access coding more easily with a voice-based interface. The ability to create code using their voice can be empowering for students with upper limb limitations, allowing them to pursue their knowledge in software development with more assurance and confidence. This tool can remove this barrier by offering a voice-based coding interface, allowing students with upper limb limitations to engage in coding activities and will even encourage them to pursue careers in software development more successfully and quickly. This is a novel approach to the issue of inclusion in the world of software development because conventional coding environments have relied on possibly unjustified presumptions about input methods and motor abilities. We can level the playing field and guarantee that people with upper limb limitations have equal access to opportunities in software development by developing tools that accepts voice input to provide accessibility.

1.7. Research Challenge

The identified challenges are listed below for the research period.

Table 1: Research Challenge

Challenge	**Justification**
Data collection and curation	Collecting and curating a high-quality dataset will accurately reflect the task at hand. creating a dataset that contains natural language voice commands and corresponding Quorum source code snippets would have been a significant challenge
Domain-specific language modeling	Existing language models are not optimized for Quorum-specific syntax and structure. The model should be trained to understand the voice commands and generate the relevant code block. It might be time consuming to train with better accuracy.
Complete all features of the tool within the given time constraint.	It is complicated to build all the features of a tool from the given time constraint due to time consumption in understanding the technologies necessary to develop the tool.
Building the necessary models and integrating it with the tool.	It takes time to learn the new techniques used for voice recognition and build the models. The models should be able to integrate with the tool developed.
Experiencing and testing the tool as a normal programmer.	Sensing the requirements while implementing the tool appropriate to the syntax created as a normal programmer is difficult.
Finding people with disability and conducting the evaluation of the tool	It is challenging to find and conduct the evaluation with consent to get proper feedback on the created mobile development language syntax and tool.

1.8. Research Questions

RQ1: How far can voice-based programming replace the conventional keyboard and mouse-based programming?

RQ2: What are the barriers and problems faced by the students with upper limb disability in learning programming concepts and accessing programming tools?

RQ3: What language should be considered for the source code generation for the differently abled students to easily learn programming?

RQ4: What language model should be used to train with the custom dataset to generate the domain-specific language syntax?

1.9. Research Aim

The aim of this research is to design, develop and evaluate a voice assisted tool for source code generation for students with upper limb disability.

To further elaborate on the aim, a tool will be developed which will generate Quorum programming language source code for natural language voice commands for the upper limb disabled students to learn programming. The tool will allow the user to give general language voice input and generate the relevant quorum language source code. The system will be designed using appropriate research expertise in the field and standard procedures at all stages.

1.10. Research Objectives

Table 2:Research Objectives

Research Objectives	**Description**	**Learning Outcomes**	**Research Questions**
Problem Identification	Conduct detailed research to identify a potential problem that needs to be resolved in the selected domain. • **RO1:** To research on the existing problems from the chosen research and problem domains.	LO2, LO3	RQ1, RQ2, RQ3

	• **RO2:** To research on the existing techniques and approaches in the chosen research and problem domains.		
Literature Review	Analyze and attain the following areas by conducting in-depth research. • **RO1:** To research on the problems and limitations faced by students with upper limb disability to learn programming. • **RO2:** To research on barriers in accessing current tools to learn programming for students with upper limb disability. • **RO3:** To research on existing voice-based tools created for students with different disabilities. • **RO4:** To research on other existing programming languages and simple syntaxes created for differently abled students. • **RO5:** To research on existing natural language to code generation approaches and techniques. • **RO6:** To identify the techniques in the existing systems and start gaining knowledge in the necessary techniques.	LO1, LO4, LO5	RQ1, RQ2, RQ3, RQ4
Requirement Elicitation	Taking the appropriate action to collect user requirements for the following areas. • **RO1:** To gather information on requirements of a voice-based code generation tool for upper limb disabled students to learn programming.	LO1, LO3, LO4, LO5, LO6	RQ1, RQ2, RQ3

	• **RO2:** To gather the requirements and expectations from stakeholders and researchers in order to understand how they will expect it to work. • **RO3:** Gather information on barriers and difficulties in accessing the tools for upper limb disabled students. • **RO4:** Gather information on the complexity of the syntax expected by the upper limb disabled students with ethical consent. • **RO5:** To gather information on the problems faced by the end users with legal consent.		
Design	Developing the designs of the proposed tool and syntax • **RO1:** To design the key components of the voice recognition tool by figuring out the essential features and helpful elements that would benefit the students to learn programming. • **RO2:** To design the elements and features of the voice-based tool for students with upper limb disability. • **RO3:** To design the necessary models to develop the code generation tool.	LO3, LO4	RQ1, RQ2, RQ3
Implementation	Developing the prototype with the necessary software resources in order to meet the requirements and design criteria outlined above	LO3, LO4, LO5, LO7	RQ1, RQ2, RQ3

	• **RO1:** To implement the voice recognition section of the tool. • **RO2:** Train the core component which is the code generation model with appropriate language model. • **RO3:** To integrate the voice recognition section with the code generation model. • **RO4:** To develop all the components of proposed tool with web UI.		
Testing and Evaluation	Evaluation and testing of the prototype • **RO1:** To perform the functionality and non-functionality testing. • **RO2:** To evaluate and examine the tool's usability and efficiency. • **RO3:** To evaluate the model's accuracy on generating and editing codes for the user voice commands. • **RO4:** To evaluate the tool and get feedback from the end users with ethical clearance. • **RO5:** To produce a comprehensive report for the academic and scientific community.	LO4, LO6, LO7, LO8	RQ1, RQ2, RQ3, RQ4

1.11. Project Scope

The scope of the project is defined below depending on the objectives, findings, existing work, and available time.

1.11.1. In-scope

- English language-specific input commands
- Natural language commands to Quorum source code generation

- Variations allowed in giving natural language instructions
- Types and Variables and control variables were covered for multiple variations and only the basic classes and actions structure is included

1.11.2. Out-scope

- Navigation support by voice commands
- Classes and actions advanced syntax structures are not generated
- Multiple variations are restricted for classes and actions
- Multi language-specific input commands

1.11.3. Prototype Feature Diagram

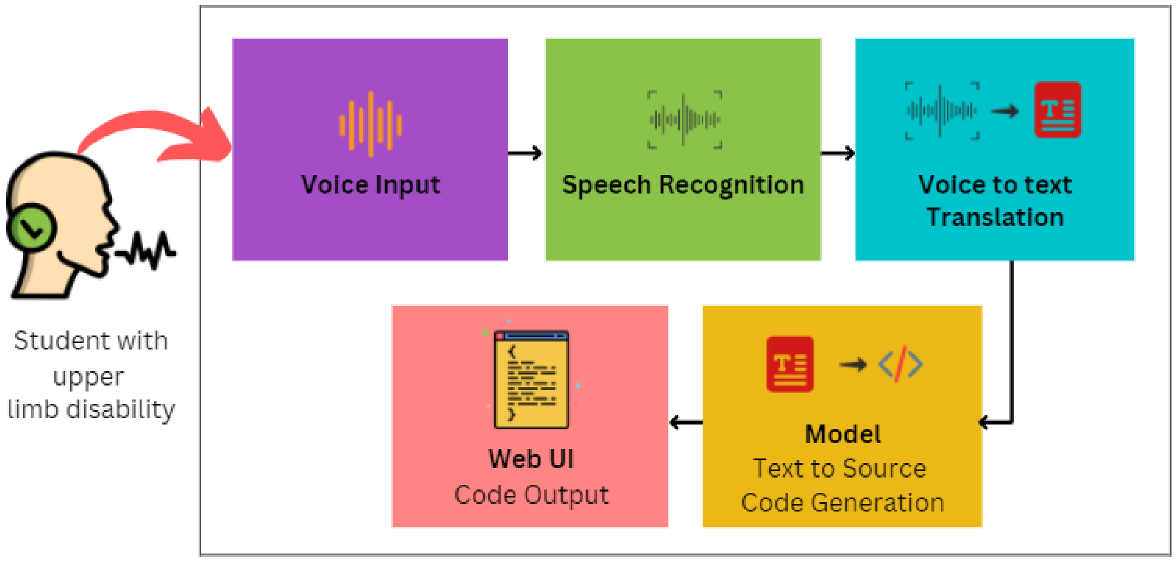

Student with upper limb disability
Voice Input
Speech Recognition
Voice to text Translation
Model
Text to Source Code Generation
Web UI
Code Output

Figure 1:Prototype Feature Diagram

1.12. Chapter Summary

The research focuses on the barriers in learning programming as students with upper limb disabilities. A voice assisted code blocks generating tool for general language corpus with error fixing using voice commands using AI is proposed as a solution to overcome this problem. The problem was discussed in this chapter together with the necessary reasons, a description of the pertinent field, the research gap, and obstacles, as well as the targets and objectives.

CHAPTER 2: LITERATURE REVIEW

2.1. Chapter Overview

The chapter provides an in-depth analysis of existing research and knowledge in the field of study. This chapter includes the concept map to identify key concepts and relationships in the literature and provide a framework for organizing and presenting the literature review, discussion of the problem domain, existing work related to the research topic, including the most relevant research papers with the summarization of the key findings, review and analysis of the current technological landscape of the research topic, including the tools, techniques used and the review on evaluating and validating the approaches.

2.2. Concept Map

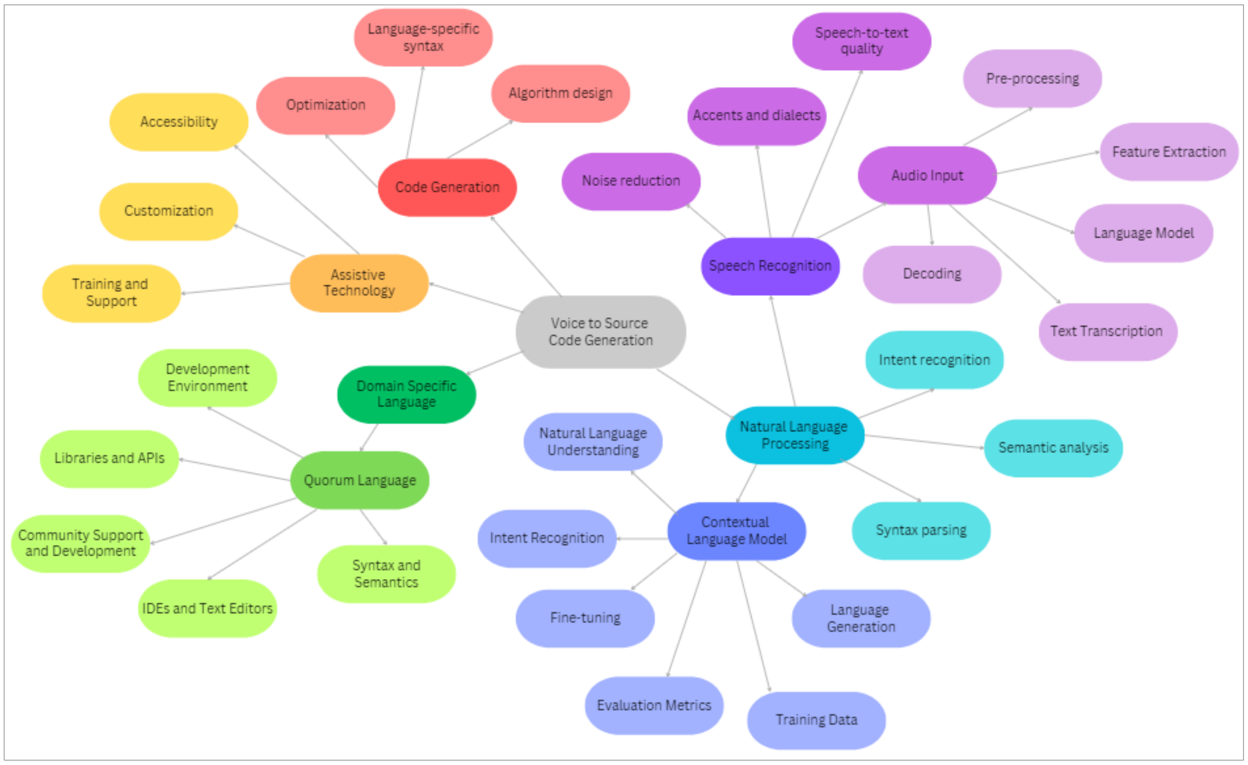


Figure 2: Concept Map

2.3. Problem Domain

Natural language commands to code generation involves the challenge of transforming human language, which is inherently ambiguous and complex, into machine-readable code that can be executed by a computer. This domain is motivated by the need to make programming more accessible to non-experts, including individuals with disabilities who may have difficulty typing or reading written code. The use of natural language commands for code generation can potentially reduce the time and effort required for programming and increase the accessibility of the skill to a wider audience. The use of natural language commands for code generation has the potential to make programming more accessible and appealing to students with disabilities, who may face barriers to traditional programming methods to help to spark an interest in programming.

2.3.1. Upper Limb Disability

Upper limb disability is a condition that affects millions of people worldwide, leading to functional limitations in the arms, shoulders, and hands. This disability can be caused by a variety of factors, including injury, disease, or congenital disorders, and can have a significant impact on an individual's quality of life (Ilse Lamers, 2015).

Upper limb disability can be caused by a variety of factors, including traumatic injuries, repetitive strain injuries, neurological conditions, and congenital abnormalities. Traumatic injuries such as fractures, dislocations, and tendon injuries are common causes of upper limb disability, often resulting from accidents or sports-related injuries. Repetitive strain injuries, such as carpal tunnel syndrome and tennis elbow, are also common among individuals who engage in repetitive tasks that involve the use of the upper limbs. Neurological conditions such as stroke, cerebral palsy, and multiple sclerosis can also cause upper limb disability, resulting in muscle weakness, spasticity, and coordination problems. Congenital abnormalities such as missing or underdeveloped limbs can also cause upper limb disability (Davidson, 2004).

2.3.2. Programming and Upper Limb Disabled Students

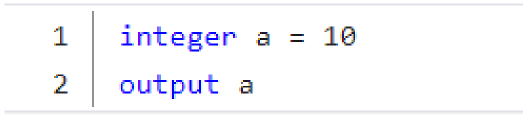
Learning programming is becoming increasingly important in today's technology-driven world. However, students with upper limb disabilities may face challenges in learning programming due to the reliance on keyboard and mouse input. The improvement of inclusion, effectiveness of accessibility, and equity in education all depend on the support provided to students with disabilities (Miyazaki, 2019). The population is growing and technical advancements in the programming sector are accelerating, yet because of their academic needs, children with disabilities require accommodations and assistive technology. This restriction makes it impossible for this demographic to work in the programming sector (F. H. Boot, 2018). Students with upper limb limitations have a number of difficulties when attempting to learn programming despite the significant advancements that have occurred recently (Lagergren, 2021).

The ineffectiveness of the accessibility, lack of diversity, and inadequate support provided by technologies and tools for learning programming are the main obstacles to be worried about. For students with upper limb limitations, using a normal mouse and keyboard to use these programming tools is difficult because typing is a physical act (Ludi, 2022). Most computer applications and settings are designed to allow users to interact with them using hand and finger gestures. Students with upper limb limitations may find it difficult to learn programming in an environment that depends on the dexterity of human hands (Cody L. McDonald, 2019). Students with the aforementioned impairment are unable to access the highly integrated programming tools using a mouse and keyboard in a traditional manner (F. H. Boot, 2018).

2.3.3. Quorum Programming Language

The Quorum Programming Language was created to make programming easier for beginners, especially for individuals with disabilities. The object-oriented, statically typed language Quorum offers benefits like soundness, usability, and scalability. Differently abled programmers will find Quorum particularly approachable due to a number of its features. For instance, it has a screen reader that can read out the code as it is being produced. Additionally, Quorum features a streamlined syntax that makes it simpler to learn and use, especially for new programmers. Advanced features like database access and object-oriented programming capabilities are also included in Quorum. It can be used to make a wide range of applications, including games, instructional software, and scientific simulations.

One unique aspect of Quorum is its emphasis on evidence-based programming. This means that the language is designed to incorporate research findings about what programming practices are most effective for different types of users. Integer a with value 10 and printing the variable is defined as shown in the snippet below in the Quorum language.



```
integer a = 10
output a
```

Figure 3: Quorum Code snippet

Quorum is used primarily in educational settings, particularly in introductory programming courses since it has easy syntax to easily understand the concepts. It has been shown to be effective in helping students with disabilities learn programming concepts and has also been used to teach programming to individuals without disabilities as well.

2.3.4. Code generation

Code generation is the process of creating computer program source code automatically from a high-level specification or input. In general, by automating software development, code generation aims to cut down on the time and labor needed to create dependable, effective code (E. Dehaerne, 2020). There are several methods for creating code, including template-based generation, model-driven generation, and domain-specific language-based generation. Domain-specific natural language-based generation involves converting natural language commands for a particular domain and generating code from that language. Widely used programming languages like Java, Python, C#, C++, Ruby, Typescript have tools for voice to source code generation to do advanced coding (S. Hossain, 2021). For upper limb disabled students to learn programming concepts and get understanding in language syntax with accessibility barriers to code by hands is considered in this research to provide solution. Based on the research currently no tool available for code generation that can directly convert natural language voice instructions into Quorum syntax.

2.3.5. Natural Language Processing (NLP)

Natural Language Processing is a branch of artificial intelligence that deals with the interaction between computers and human language. In order to carry out numerous tasks, it entails processing and comprehending natural language data, including text, speech, and voice instructions (Yue Kang, 2020).

Natural Language Voice Commands are a particular kind of natural language data that users speak to communicate with tools, programs, or services. Voice commands can be used for many different things, including setting reminders, playing music, placing calls, and managing smart home appliances. With the aid of speech recognition technology, natural language voice commands may be translated into text, which can then be processed with NLP techniques to glean pertinent data and produce relevant answers (Prakash M Nadkarni, 2011).

2.3.6. Natural Language Commands to Code Generation

The majority of individuals are more accustomed to and intuitive with natural language requests. With less concern about the exact syntax and organization of the programming language, they enable newcomers to communicate their intentions in a more natural way. Instead of needing to learn the syntax for creating and assigning variables in a specific programming language, a novice may instead say, "create a integer variable called myNumber and assign value 5". The generation of computer code from natural language commands can offer a more user-friendly and intuitive interface for learning programming, allowing newcomers to concentrate on comprehending the logic and concepts behind programming rather than becoming mired down in syntax and structure (Prakash M Nadkarni, 2011).

2.4. Existing Work

Natural language voice command to source code generation is a relatively new and emerging area of research, and there have been several recent works exploring this problem.

2.4.1. Generative AI and code generation

Generative AI is closely connected to code generation because both techniques involve creating new data or code based on existing patterns or rules. In the case of generative AI, the model is trained on a large dataset and then learns to generate new data that is similar to the original dataset. This could include generating images, music, or even text. One application of generative AI in code generation is the use of natural language processing (NLP) models to generate code from plain language specifications (Yuding Liang, 2018). This approach involves training a model on a large dataset of code examples and natural language descriptions of what the code should do. The model then learns to generate code based on the given specifications. Another example of generative AI in code generation is the use of GANs to generate new code examples. In this approach, the generator network is trained to create new code that is similar to the existing dataset, while the discriminator network tries to distinguish between the generated code and the real code. Through an iterative process, the generator becomes better at creating code that is indistinguishable from real code. Overall, generative AI has the potential to revolutionize code generation by enabling faster and more efficient development of software systems (Lili Mou, 2015).

2.4.2. Rule-based systems for code generation

Rule-based systems are an example of an expert system, which is a type of AI system designed to solve problems in a specific domain using a set of rules. CodeSynthesis is a code generation tool that uses a set of predefined rules to generate C++ code based on a given schema or specification. CodeCharge Studio is a web development tool that uses a set of predefined rules to generate code for various web applications based on user input. Ruby on Rails is a popular web development framework that includes a scaffolding tool for generating code based on predefined rules and templates. XSLT is a rule-based language for transforming XML documents, and can be used for generating code in various programming languages. Model-driven engineering (MDE) tools use a set of predefined rules to generate code based on high-level models or specifications (E. Dehaerne, 2020).

While rule-based systems can be effective for generating code in specific scenarios, they can be limited in their flexibility and ability to handle complex requirements. Machine learning-based and hybrid systems are often used in combination with rule-based systems to overcome these limitations and provide more accurate and adaptable code generation solutions.

2.4.3. Machine learning-based systems for code generation

Machine learning-based systems are a type of AI system that use statistical algorithms to learn patterns in data and make predictions or decisions. There are several machine learning-based systems for code generation that have been developed in recent years. DeepCoder is a system developed by researchers at Microsoft that uses machine learning techniques to automatically generate code from natural language descriptions of a program's desired behavior. CGAL is a machine learning-based system developed by researchers at Carnegie Mellon University that generates code using a combination of a neural network and a rule-based system (E. Dehaerne, 2020). CodeTrans is a machine learning-based system developed by researchers at Facebook AI that can translate code between different programming languages. Tree-to-Tree Neural Machine Translation (T2T-NMT) is a machine learning-based system developed by researchers at Google that can translate between different programming languages by treating code as a natural language. CodeBERT is a machine learning-based system developed by researchers at Microsoft Research Asia that uses transformer-based models to learn patterns and relationships between natural language and code and can generate code from natural language descriptions.

These systems use different approaches to machine learning for code generation, ranging from neural networks to transformer-based models. They have shown promising results in generating code that is accurate and efficient, but they also have their own limitations and require large amounts of data and computational resources for training (E. Dehaerne, 2020).

2.4.4. Hybrid systems for code generation

Hybrid systems combine rule-based and machine learning-based approaches to achieve better performance. They may use pre-defined rules to guide machine learning algorithms or use machine learning to identify new rules that can be added to the system. Neural Sketch Learning (NSL) is a hybrid system developed by researchers at MIT that combines a neural network with a rule-based system to generate code for programming problems. DeepCoder+ is an extension of the DeepCoder system developed by researchers at the University of Cambridge that combines neural networks with symbolic execution to generate code that is both accurate and efficient. ML4Code is a hybrid system developed by researchers at Microsoft that combines machine learning with program analysis to generate code that is both accurate and safe. CodeHint is a hybrid system developed by researchers at the University of Illinois that combines machine learning with expert knowledge to generate code hints for developers (E. Dehaerne, 2020).

These hybrid systems combine the strengths of both rule-based and machine learning-based approaches to generate code that is accurate, efficient, and safe. They have shown promising results in generating code for a variety of programming languages and applications, but they also require careful design and integration of both approaches to achieve optimal performance.

2.4.5. Domain-specific approaches for code generation

Domain-specific approaches are designed to solve problems in specific fields or industries. They use specialized algorithms or techniques that are tailored to the needs of the domain. CodePhage is a domain-specific approach for generating code that involves analyzing existing code to identify common patterns and then using those patterns to generate new code. It has been used to generate code for web applications and mobile applications. CodeGenie is a domain-specific approach for generating code that uses natural language processing techniques to understand user requirements and generate code that satisfies those requirements. It has been used to generate code for database applications and web applications (E. Dehaerne, 2020).

These domain-specific approaches for code generation can be highly effective when used in the appropriate context. They can generate code that is more accurate, efficient, and easier to maintain than code generated using more general-purpose techniques. However, they also require specialized knowledge and expertise to develop and use effectively.

2.4.6. Augmented Intelligence for code generation

Augmented intelligence refers to systems that use machine learning to augment human intelligence. These systems assist humans in making decisions or performing tasks, rather than replacing them entirely. Augmented intelligence systems fall under the broader category of AI systems. Kite is an AI-powered code completion tool that uses machine learning to suggest code snippets and corrections to developers as they write code. Kite's algorithm learns from millions of code examples to provide personalized and accurate code suggestions. Codota is an AI-powered code completion tool that uses machine learning to suggest code snippets and corrections to developers as they write code (Yuding Liang, 2018). Codota's algorithm learns from millions of code examples to provide personalized and accurate code suggestions. Deep TabNine is an AI-powered code completion tool that uses deep learning to suggest code snippets and corrections to developers as they write code. Deep TabNine's algorithm learns from millions of code examples to provide personalized and accurate code suggestions. GitHub Copilot is an AI-powered code completion tool that uses machine learning to suggest code snippets and corrections to developers as they write code. GitHub Copilot's algorithm learns from millions of code examples to provide personalized and accurate code suggestions. TabNine is an AI-powered code completion tool that uses machine learning to suggest code snippets and corrections to developers as they write code. TabNine's algorithm learns from millions of code examples to provide personalized and accurate code suggestions (E. Dehaerne, 2020).

These tools can be used to assist developers in writing code, but they do not generate complete code from natural language commands like some of the other approaches we have discussed. Instead, they provide suggestions and corrections based on code that has already been written.

2.4.7. Transfer Learning for code generation

Transfer learning is a machine learning technique that involves training a model on one task and then transferring that knowledge to another related task. Transfer learning is a subfield of machine learning. CodeBERT is a transformer-based language model that has been specifically designed for source code tasks, including code generation. It has been trained on a large dataset of code and natural language to learn patterns and relationships between them. GPT-3 is a state-of-the-art transformer-based language model that can be fine-tuned for code generation tasks (M. T. Tausif, 2018). It has been used to generate code for a variety of programming languages, including Python, JavaScript, and HTML/CSS. T5 (Text-to-Text Transfer Transformer) is a transformer-based language model that can be fine-tuned for a variety of NLP tasks, including code generation. It has been used to generate code for a variety of programming languages, including Python, JavaScript, and SQL. RoBERTa is a transformer-based language model that has been pre-trained on a large corpus of text data, including code. It can be fine-tuned for code generation tasks, including generating Python code from natural language commands. Transformer-XL is a transformer-based language model that is designed to handle long-term dependencies in text data. It has been used for code generation tasks, including generating Python code from natural language commands (E. Dehaerne, 2020).

2.4.8. Combination of Transfer learning and Machine learning-based systems

Transfer learning can be used to improve a language model that has already been trained to generate source code from commands in natural language. For instance, to create code from new natural language instructions, a pre-trained transformer model like GPT-3 might be adjusted using a dataset of natural language commands and accompanying source code.

Finetuning a pre-trained T5-small model with a custom dataset involves transfer learning. Transfer learning is a machine learning technique that involves leveraging knowledge learned from a pre-trained model (in this case, T5-small) and applying it to a new, related task. During finetuning, the pre-trained T5-small model is further trained on a smaller dataset specific to the target task to improve its performance on that task. So, finetuning a T5-small model with a custom dataset involves both transfer learning and machine learning-based systems. This will be the suitable approach for this research to develop the source code generation tool (E. Dehaerne, 2020).

Following are the currently existing tools which generates source code for widely used programming languages by understanding natural language prompts but none of the currently available tools can generate code for the quorum programming language.

OpenAI Codex: Several programming languages, including Python, JavaScript, and Ruby, can be used to create code using the OpenAI Codex natural language processing model. To produce code from natural language inputs, Codex combines machine learning and heuristics. It is trained on a sizable corpus of existing code to increase its accuracy.

Snips: Snips is a platform for building custom voice assistants that can generate code in a variety of programming languages. Snips uses a natural language understanding (NLU) engine to parse natural language inputs and generate code in real-time.

Serenade AI: Real-time code generation enabled by AI in an editor that can interpret natural language commands. By allowing people to employ everyday coding operations like declaring variables, calling functions, and navigating code in normal language, it is intended to assist developers in writing code more quickly and effectively. To comprehend inputs in natural language and produce code, Serenade AI combines machine learning models and rule-based systems. The technology learns common patterns and correlations between natural language and code through training on a sizable dataset of code snippets and natural language inputs.

GitHub Copilot: AI-powered tool developed by GitHub and OpenAI for code completion. It interprets inputs in natural language and generates real-time code suggestions using machine learning models. Copilot is constructed on top of the OpenAI GPT (Generative Pre-trained Transformer) language model, which has been trained using a sizable corpus of natural language and programming code.

2.4.9. Summary of Existing work

The summary of existing work in the similar research areas are listed in the following table.

Table 3: Related Work

Citation	Technique	Improvement	Limitations
(M. G. Prakash, 2022)	• Stemming, lemmatization, and tokenization are employed in an NLP	With the use of wave forms, text to speech content was produced with excellent quality and assurance. The code	• Limited programming languages for source code generation specifically Quorum

	model to extract the necessary keywords • The code generator will produce the necessary code snippets using the keywords.	generator is connected to the syntax library, which has syntax for several programming languages. The relevant code snippets are generated and sent as responses.	language is not included which is an evidence based language created for differently abled students to learn programming.
(Nizzad, 2022)	• Transformer model architecture • Tokenization of source and target • Text Auto Completion and Source Code Generation • Encoder-Decoder Framework in Deep Sequence Modeling	Determining the most effective algorithm for text-to-source-code creation which would satisfy the study's completeness requirements and offer a fresh way to build any speech-to-text or speech-to-speech applications.	• There is only a corpus of general language available. A corpus of programming languages is not created. • Simple syntax programming languages like quorum are not considered for voice programming.
(Adnan Rahim Khan, 2022)	• Voice to text • Text to source code generation	A voice-based graphical user interface and an integrated development environment (IDE) to learn and use programming concepts for the people with physical disabilities.	• Wrong representation of words and mistakes due to complex syntax • Quorum programming language is not considered for source code generation.

(Bharathi, 2016)	• Speech Recognition	Voice input is used to record a C program, which is then generated into the compiled program.	• The suggested voice-based IDE is only provided for C programs. The complicated syntax is not modified. • Direct voice to text generation without natural language command understanding.
(Lili Mou, 2015)	• RNNs to learn the mapping from natural language descriptions of programming problems to source code • Beam search to generate multiple candidate programs and select the most likely one	End-to-end program generation using deep learning	• Generated code may not always be syntactically correct or fully executable. • Require additional training on new datasets to improve its performance

2.5. Technological Review

2.5.1. Transformer based language models

For natural language processing (NLP) tasks, transformer-based language models use the transformer architecture. The transformer architecture, which was created expressly for NLP tasks, is now considered to be the standard for many latest NLP models. Because the transformer architecture is based on a self-attention mechanism, the model may evaluate the relative weights of the various words in a phrase as it is processed. Due to its ability to account for long-range linguistic relationships, the model is highly suited for jobs like question-answering, text summarization, and language translation. Transformer-based language models are pre-trained on large amounts of text data using unsupervised learning, which allows the models to learn general language patterns and structures. Once pre-trained, the models can be fine-tuned on specific NLP tasks using supervised learning, which allows the models to learn task-specific features and improve their performance on the target task. Most suitable language model can be used to generate the Quorum language source code for natural language commands. This will be reviewed in detail under the technological review section (Prakash M Nadkarni, 2011).

The following table provides the summary and review of existing transformer based language models available.

Table 4:Langauge model review

Language Model	Description	Discussion
GPT-3:	GPT-3 (Generative Pre-trained Transformer 3) is a state-of-the-art language model developed by OpenAI. It has been used for a wide range of natural language processing tasks, including code generation. GPT-3 can generate code for various programming languages, such as Python, JavaScript, and HTML/CSS.	Largest transformer-based language models with up to 175 billion parameters. The model consists of multiple layers of transformer blocks, making it highly complex and computationally expensive.
CodeBERT	CodeBERT is a transformer-based language model that has been specifically designed for source code tasks, including code generation. It has been trained on a large dataset of code	Relatively smaller model to GPT-3 size of 12-384 million parameters depending on the variant used. Despite its smaller size, CodeBERT has shown impressive results on several code generation tasks.

	and natural language to learn patterns and relationships between them.	
Transformer-XL	Transformer-XL is another language model that can be used for code generation. It is a variant of the original Transformer model that is specifically designed for longer sequences of text. It has been shown to perform well on a variety of natural language processing tasks, including code generation.	Has a model size of up to 257 million parameters. It uses a unique segment-level recurrence mechanism that allows it to handle longer sequences of text data and has shown promising results on code generation tasks
RoBERTa	RoBERTa (Robustly Optimized BERT Pretraining Approach) is a language model that is based on the BERT architecture. It has been trained on a large corpus of text and can generate code for various programming languages.	Has a model size of up to 355 million parameters and has been trained on a large corpus of text data, including code. It can be fine-tuned for code generation tasks, including generating Python code from natural language commands.
T5	T5 (Text-to-Text Transfer Transformer) is a language model developed by Google that can be used for a wide range of natural language processing tasks, including code generation. It has been trained on a massive dataset of text and can generate high-quality code snippets for various programming tasks.	Has a model size ranging from 60 million to 11 billion parameters, depending on the variant used. It uses a "text-to-text" approach to NLP tasks and has been used to generate code for several programming languages.
Gshard	Gshard is a transformer-based language model that has been designed for training very large models using distributed training. It can be used for	Has a model size of up to 600 billion parameters and is designed for training very large models using distributed training. It can be used for

	code generation tasks, including generating Python code from natural language commands.	code generation tasks and has shown promising results in several experiments.

2.5.2. T5 (Text-to-Text Transfer Transformer)

T5 (Text-to-Text Transfer Transformer) is a transformer-based language model developed by Google that is capable of performing a wide range of natural language processing tasks. T5 is unique in that it uses a single architecture and training procedure for a variety of tasks, which makes it highly flexible and efficient (Colin Raffel, 2020).

There are several variants of the T5 model that have been developed for specific tasks or to improve its performance. Here are some of the most notable T5 variants:

1. **T5-Small**: This is a smaller variant of the T5 model with 60 million parameters, which is significantly smaller than the original T5-Base model with 220 million parameters. Despite its smaller size, T5-Small has shown promising results on several NLP tasks, including question answering, summarization, and code generation.
2. **T5-Base**: This is the original T5 model with 220 million parameters. It is trained on a large and diverse corpus of text data and can be fine-tuned for a variety of NLP tasks, including text classification, summarization, and translation.
3. **T5-Large**: This is a larger version of the T5 model with 770 million parameters. It is designed to handle more complex natural language tasks and has been shown to outperform T5-Base on several benchmarks.
4. **T5-XXL:** This is the largest variant of the T5 model with 3 billion parameters. It is designed for tasks that require a high level of precision and accuracy, such as language modeling and translation.

The T5-small model is a transformer-based language model that has been trained on a large and diverse corpus of text data, making it a powerful tool for natural language processing tasks such as generating code from natural language commands. It is also relatively lightweight compared to larger T5 models, making it more suitable for fine-tuning on smaller custom datasets (E. Dehaerne, 2020).

In addition, the T5 model architecture is designed for transfer learning, which allows it to leverage its pre-trained knowledge to adapt to new tasks with minimal additional training. This makes it a great candidate for fine-tuning on a custom dataset of natural language commands and their corresponding Quorum language source code.

2.6. Evaluation and benchmarking

The evaluation of a transformer based language model is necessary to assess the model's ability to perform its intended task accurately and effectively. A language model is usually trained to predict the probability distribution over a sequence of words given a previous sequence of words. In this research for a natural language voice commands, it should be able to generate the relevant source code. The language model is finetuned with custom dataset to generate Quorum language source code (Joel Jang1, 2023).

An appropriate custom evaluation dataset which contains natural language commands along with the corresponding Quorum language source code is necessary to check if the model properly generates the correct source code for the trained variations and new set of variations needs to be considered as well in the validation dataset. This evaluation dataset should contain a diverse set of natural language voice commands along with the corresponding Quorum language source code. The evaluation dataset should be designed to cover a wide range of scenarios and variations to ensure that the model can generalize well to new input. It is important to consider new variations and scenarios in the validation dataset to ensure that the model can generalize well to new inputs.

Evaluating a transformer-based language model requires a combination of quantitative and qualitative analysis. The evaluation metrics used to assess the performance of the transformer-based language model should also be chosen carefully based on the specific task. For example, the primary evaluation metric may be the accuracy of the generated source code, which measures how often the model correctly generates the expected output for a given input. Also evaluate metrics such as accuracy, precision, recall, F1-score needs to be measured to understand the model's performance on specific tasks (Joel Jang1, 2023).

The following table provides the detailed description and objective orientation of the evaluation metrics.

Measure	Description	Objective Orientation
Perplexity	This is a measure of how well the model predicts the next word in a sequence. It is calculated as the exponentiation of the cross-entropy loss and is often used for language modeling tasks.	Higher value is preferable.
Accuracy	This measures the proportion of correct predictions made by the model on a given dataset. It is commonly used for classification tasks.	
Precision and recall	These are used for binary or multiclass classification tasks. Precision measures the proportion of true positives among the predicted positives, while recall measures the proportion of true positives among all actual positives in the dataset.	
F1-score	This is the harmonic mean of precision and recall and provides an overall measure of a model's performance in classification tasks.	
Mean squared error (MSE)	This is commonly used to evaluate regression models and measures the average squared difference between the predicted and actual values.	Lower value is preferable.
Mean absolute error (MAE)	This is also used to evaluate regression models and measures the average absolute difference between the predicted and actual values.	

BLEU (Bilingual Evaluation Understudy): BLEU is a metric used to evaluate the quality of machine-translated text by comparing it to one or more human translations. The BLEU score ranges from 0 to 1, with a higher score indicating a better match between the machine-generated text and the human-generated text. The BLEU score is calculated as follows:

BLEU = BP * exp(sum(wi * log(precisioni)))

where BP is the brevity penalty, wi is the weight assigned to the i-gram, and precisioni is the precision of the i-gram.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation): ROUGE is a set of metrics used to evaluate the quality of summarization systems by comparing them to one or more reference summaries. The ROUGE score ranges from 0 to 1, with a higher score indicating a better match between the system-generated summary and the reference summary. The ROUGE score is calculated as follows:

ROUGE = (ngram match count) / (total ngrams in reference summary)

where ngram match count is the number of n-grams that appear in both the system-generated summary and the reference summary, and total ngrams in reference summary is the total number of n-grams in the reference summary.

2.6.1. Holdout Validation

In holdout validation, the dataset is split into training and testing sets, and each model is trained on the training set and evaluated on the testing set. The performance metrics of the selected models are then compared to determine which one performs better. There are various language models that can be used for this similar task to generate code, but Quorum language code generation is not available with existing models. According to the literature review T5 model is selected for this research for code generation and the discussion and reasonings are included in this chapter. Due to the limitation of unavailable benchmarking standards, the selected evaluating approach is to compare the T5 variants models and test and evaluate the selected variant.

2.7. Chapter Summary

This chapter discusses the existing works related to code generation with natural language commands and the approaches made so far related to this research. An in-depth review was done on the technical approaches related to code generation and natural language voice commands. Review on evaluating a finetuned transformer based language model to perform a specific task was included. Overall, the chapter thoroughly analyses the problem domain, existing works technical review and evaluation review and holdout validation for source code generation to natural language commands.

CHAPTER 3: METHODOLOGIES

3.1. Chapter Overview

Through the use of appropriate procedures, projects are standardized. Definition of methodologies is necessary to keep the project's quality high. In this chapter, the approaches used at various stages of the research are discussed under the subtopics research, development and project management methodology.

3.2. Research Methodology

Table 5:Research Methodology

Aspect	**Justification**
Research Philosophy	Among positivism, pragmatism, realism, and interpretivism, the **pragmatism approach** was chosen because the authors will examine and experiment with multiple methodologies as an element of a combined strategy.
Research Approach	The **deductive approach** has been chosen since the project moves from a hypothesis to validating it. Both qualitative and quantitative methodologies were chosen as the strategy for data analysis since the interview contained both closed and open ended.
Research Strategy	In order to respond to the stated research questions, strategy is crucial. Informal interviews are selected as the main strategy. As a method for gathering information from stakeholders, semi-structured interviews are chosen. Observation and brainstorming also be considered to support the identified issue and the gap that needs to be addressed.
Research Choice	Between mono, mixed, and multi-method procedures, the **mixed method** was selected for this study since both quantitative and qualitative data, like interview responses containing closed and open ended questions for this research.

Research Time Horizon	The duration of the investigation is determined by the time horizon. The **cross-sectional time horizon** was chosen out of the two time horizons since the data for this project will be acquired throughout the requirement analysis phase of the research.
Research Techniques and procedures	Techniques including observations, documents, conversations, evaluation reports, interviews, and questionnaires will be used to analyze and gather data.

3.3. Development Methodology

For the research project, the **prototype model** was chosen from among the development models. Prototyping is a feasible model for those large and complicated systems that lack a manual process or existing system for determining the requirements. The development of prototypes is based on currently known requirements. The system's real feel is conveyed through the prototype. It aids in a better comprehension of the required system requirements. User feedback can be obtained in the initial stages of the project, can make changes to the finished product, errors can be detected in earlier stages than other methodologies, lacking functionalities can be identified easily, and reduces expenses were the reasons to select this development methodology among others.

3.4. Project Management Methodology

The combination model Prince2 Agile was chosen out of all the various project management approaches since it is an evolution of PRINCE2 and more responsive to change and more flexible for short-term goals and objectives to allow deadlines on time. Agile emphasizes the iterative cycle, whereas Prince2 focuses on breaking the project into achievable components.

3.4.1. Schedule

3.4.1.1. Gantt Chart

Please refer to Appendix A - Gantt Chart.

3.4.1.2. Deliverables and dates

Table 6:Deliverables and dates

Deliverable	**Date**
Project Proposal Document and Ethics form The initial proposal of the project	9th November 2022
Literature Review Document A critical, analytical overview of published research literature that is pertinent to the research issue under discussion.	27th October 2022
Software Requirement Specification Defining the project through the elicitation of software requirements.	15th January 2023
System Design Document Project and software architecture designing.	16th February 2023
Prototype Document to evaluate and test the project.	16th February 2023
Test and Evaluation Report	23rd March 2023
Thesis A dissertation that contains comprehensive project documentation.	2nd May 2023
Final Research Paper A research paper introducing the developed system at the end of this project	2nd May 2023
Review Paper A detailed and comprehensive analysis report that thoroughly and methodically assesses the available information on the topic.	15th May 2023

3.5. Resources

The resources required to accomplish the project are identified in accordance with the previously stated objectives, research techniques, and high-level architecture. The following lists the necessary hardware, software, and data resources along with explanations of their necessity.

3.5.1. Hardware Resources

- At least 16GB of RAM
- GPU with at least 8GB of VRAM
- A high-speed SSD
- minimum of 10 GB of free disk space

3.5.2. Software Resources

- Operating System - Microsoft Windows 7/8/10 (32 or 64 bit)
- Google Colab
- React
- Mendeley - Tool to manage references and save backup copies of research artifacts.
- MS Office/ Canva/ Google Docs/ Figma - Tools for producing reports, figures, and documents.
- Git Hub or Google Drive - To backup project-related files and code
- teamgantt – For Gantt chart generation

3.5.3. Technical Skills

- Research and analysis abilities
- Programming skills
- Knowledge in Deep Learning and NLP
- Knowledge of language models
- Familiarity with Google Colab or other cloud-based computing platforms
- Familiarity with transformers library

3.5.4. Data Requirements

- Custom dataset to generate Quorum source code for natural language commands.

3.6. Risks and Mitigation

The following project risks were found, and they could have an impact on the standard of the deliverables. In order for the project to achieve the requirements and be successfully completed, mitigation plans were found to address the problem.

Table 7:Risk and mitigation

Risk	**Probability of Occurrence**	**Magnitude of the loss**	**Mitigation Plan**
Specific dataset, it may become overfitted and perform poorly on other datasets.	5	5	The finetuning process was carefully monitored, and the model was evaluated on multiple datasets to ensure its generalizability.
Inadequate computational resources.	4	5	Have access to adequate computational resources, such as high-end GPUs
Finding end users to evaluate the project.	5	5	Use controlled experiment methodology for evaluation.
Lack of speech recognition efficiency	4	5	Need to create a custom unbiased dataset to train with necessary voice commands.
Health-related problems	4	5	Do overtime and reschedule the work.

Table 8: Risk and Mitigation

3.7. Chapter Summary

Pragmatism approach was chosen as research philosophy, deductive approach for research approach, informal interviews, brainstorming and observation for research strategy and mixed method as research choice for this research. Prototype model was chosen as the development model and Prince2 Agile was selected as project management methodology. Schedules, deliverable dates, risks and mitigations and the resources used for the research are also mentioned in this chapter.

CHAPTER 4: SOFTWARE REQUIREMENTS SPECIFICATION

4.1. Chapter Overview

This chapter primarily focuses on gathering requirements and analyzing the data that has been gathered. The system stakeholders are initially identified, and their roles are stated. The system-appropriate techniques for requirement gathering are then addressed. Use case diagrams and descriptions are used to explain use cases after the data has been analyzed. Finally, the functional and non-functional requirements of the prototype are described in detail.

4.2. Rich Picture Diagram

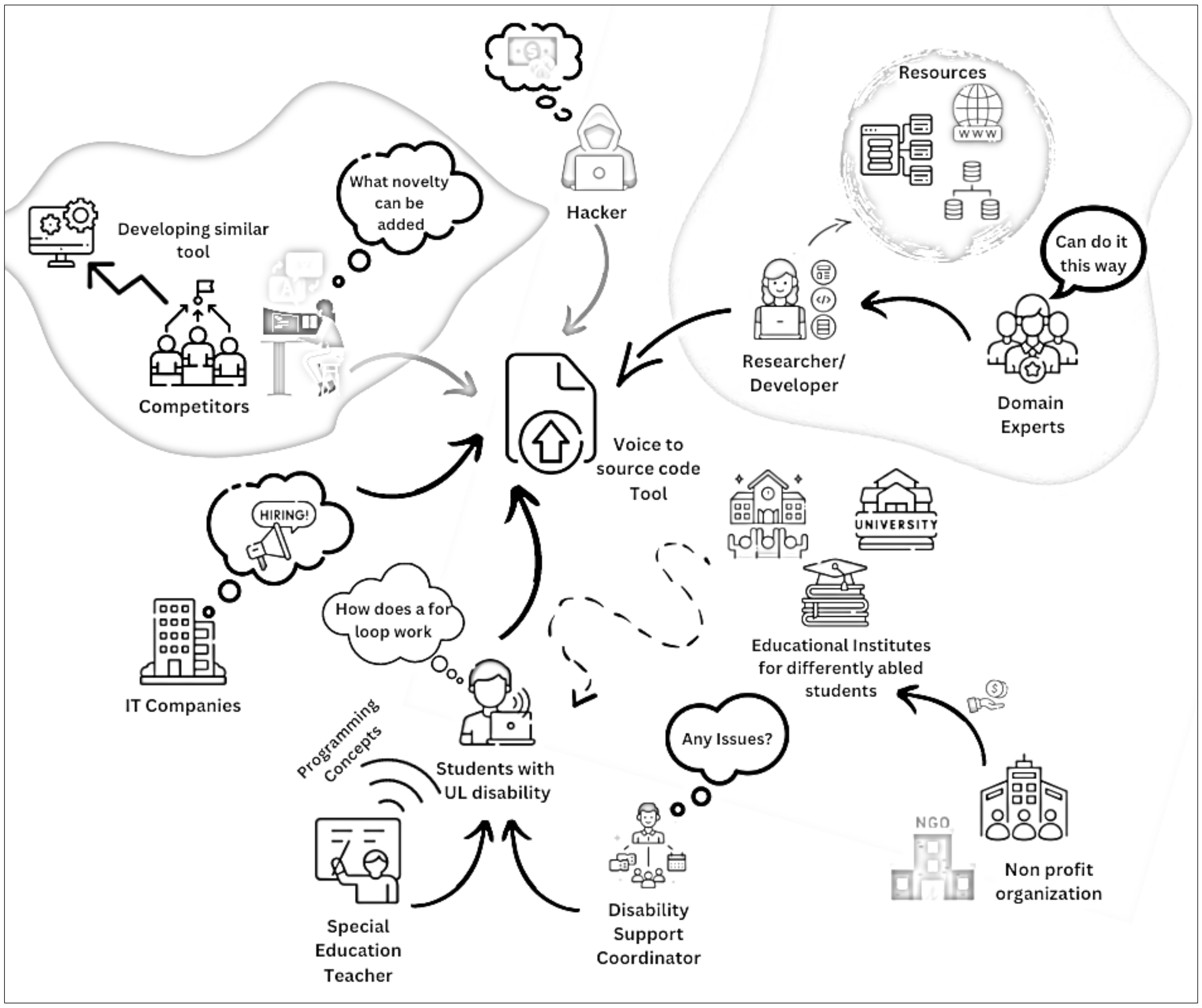


Figure 4: Rich Picture Diagram

4.3. Stakeholder Analysis

The Stakeholder Onion Model demonstrates the stakeholders connected to the system along with the stakeholder viewpoints, to describe each stakeholder's involvement in the system.

4.3.1. Stakeholder onion model

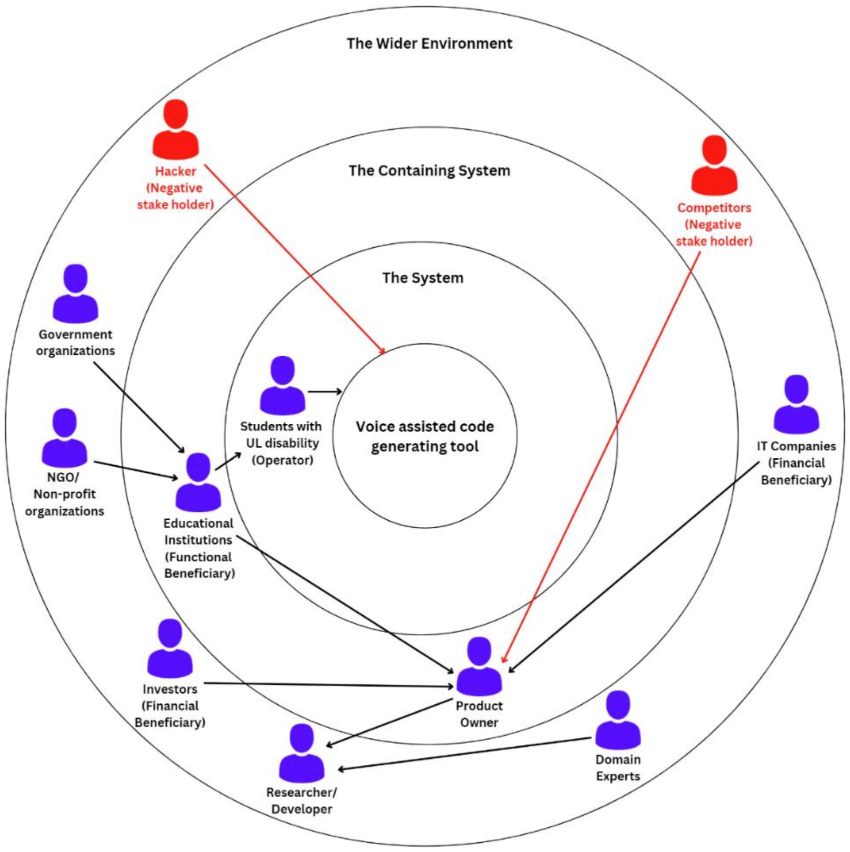

The Wider Environment
The Containing System
The System
Hacker (Negative stake holder)
Competitors (Negative stake holder)
Government organizations
NGO/ Non-profit organizations
Students with UL disability (Operator)
Voice assisted code generating tool
IT Companies (Financial Beneficiary)
Educational Institutions (Functional Beneficiary)
Investors (Financial Beneficiary)
Product Owner
Researcher/ Developer
Domain Experts

Figure 5: Stakeholder Onion Model

4.3.2. Stake holder Viewpoints

Table 9: Stakeholder Viewpoints

Stakeholder	**Role**	**Viewpoint**
Students with UL Disability	Operator	The tool will help them since it will allow them to write code more convenient without having to physically type.
Educational Institutions	Functional Beneficiary	The tool can be used as a resource by schools, colleges, and institutions that accept differently

		abled students for more inclusive learning environment.
Product Owner	System Owner (Operational Administration)	defining the tool's vision and goals and ensuring that it satisfies the requirements of the differently abled students
Researcher/ Developer	Managerial Beneficiary	By conducting research and development and communicating with the product owner, develop and improve the system.
Domain Experts	Advisory	Provides help to the developer with professional guidance and insights to enhance the features and performance of the system.
Investors	Financial Beneficiary	They can gain from the success of the product by receiving a return on their investment.
IT Companies		Companies will gain diversity of workers as a result of the growth of developers with disabilities, and IT companies will benefit financially from tax incentives for hiring them.
Government Organizations	Political Beneficiary	Governmental organizations can utilize this to enhance digital accessibility and provide equal chances for the differently abled people in the IT industry.
NGO/ Non-Profit Organizations		Nonprofit organizations support differently abled students by providing fund to the educational institutions and through assisting the students in developing new skills and enhancing their competitiveness for this industry.
Competitors	Negative Stakeholder	Tries implementing a similar tool/software by observing or illegally accessing the system
Hacker		Tries to destroy the tool or insert malware.

4.4. Selection of Requirement Elicitation

The requirements for the design and development of this research project were gathered using a variety of requirement elicitation techniques. literature review, interviews, observation, and brainstorming are the techniques selected. The justifications for choosing the designated requirement elicitation approaches are discussed below.

Table 10: Selection of Requirement Elicitation

Method 1: Literature Review
This technique is more suitable to research, compare and find existing works done and identify the gaps on the topic from the selected domain since research papers will provide the necessary details on the existing works to move further on the research. This technique focuses on comparing the systems and research that are currently available. Understanding the gaps and requirements of the existing systems is the primary priority. The study on the existing works will provide a space to come up with a new approach and solution to the identified problem.
Method 2: Interviews
To determine the best approach to resolving the issue, interviews were conducted to obtain expert perspectives on the requirements pertinent to the domain and to enhance the solution novelty through research. Initially the research papers can provide gaps and previous methods used to solve the problem, but interviews helped to get detailed knowledge of the research domain and to get advice on the novel solution that is identified for the problem.
Method 3: Observation
In order to learn about the user's demands, pain spots, and requirements, it is necessary to observe the user's behavior, activities, and working environment. The requirement gathering process benefits from observations since they help to construct a more comprehensive picture of the user's needs and requirements. Additionally, it assists in verifying presumptions made during the requirement gathering procedure and assure that the proper solution which satisfies the end-user's requirements.
Method 4: Brainstorming
To generate and identify different solutions, ideas, thoughts, and suggestions it is essential to get inputs from multiple participants related to the problem and solution domain. To come up

with as many potential solutions as you can, which can then be analyzed and improved during the requirement collection process this is the most suitable requirement gathering technique.

4.5. Discussion of Findings

4.5.1. Literature Review

Table 11: Literature Review Findings

Finding	**Citation**
Determining the most effective language model for text-to-source-code creation which would satisfy the study's complete requirements and offer a way to build voice to source code generation model for the accessible programming language for the differently abled students.	(Adnan Rahim Khan, 2022)
As a physical act of typing, it is challenging for students with upper limb disabilities to access these programming tools with conventional usage of mouse and keyboard.	(Ludi, 2022)
Programming with complex syntax consumes more time and voice strains for students. There are higher possibilities to make mistakes and for wrong representation of words while converting from voice to text.	(Nizzad, 2022)
Language models for natural language commands to source code generation. A language model is an artificial intelligence model that learns to predict the probability of a sequence of words. T5-small is the most suitable language model for natural language commands to source code generation for limited dataset.	(E. Dehaerne, 2020)
A corpus of general language commands to generate programming code blocks for simple syntax programming languages like quorum is lacking to make learning programming effective and accessible for students with upper limb disabilities.	(Ayushi Trivedi, 2018)
There are several challenges to face in the still-emerging field of speech to source code generation. One of the key obstacles is the source code's complexity, which can make it challenging to translate voice input accurately and effectively into code.	(Apostolos, 2017)

4.5.2. Interviews

Table 12: Interview Findings

Codes	Themes	Conclusion
Syntax, Code generation, voice to code generation	Barriers for students in the existing programming tools	The syntax and structure of programming languages can be challenging for students with upper limb limitations to comprehend and traverse, even with voice-to-text conversion technologies for beginners.
Quorum studio, learn programming	Inaccessible learning environment	Voice to source code generating tool for Quorum Studio for quorum programming language will provide an accessible learning environment for the differently abled students.
Vocal strain, inaccessibility to learn programming	Barriers in voice to source code generation and understanding of programming language syntax and concepts	Students who program with complicated syntax take experience vocal strain and take more time than the normal students to code by dictating each word of the syntax relevant to the programming language.

4.5.3. Observation

Table 13: Observation Findings

Criteria	Finding
Issues faced in accessing the programming tools	Voice to source code generating tool for quorum programming language will provide an accessible learning environment for the differently abled students since they find it difficult to use these programming tools with standard mouse and keyboard usage because typing is a physical act. Programming with complex syntax consumes more time and voice strains for beginner students.
Techniques and approaches	For the production of speech to source code and for source code editing tools, deep learning and natural language processing techniques are frequently utilized.

NLP for speech recognition	It is a common practice to employ natural language processing (NLP) techniques to translate processed speech input into text that may be fed into machine learning models. Named entity recognition (NER), part-of-speech (POS) tagging, and dependency parsing are examples of NLP approaches.
Transformer-based neural network model for source code generation.	Modern performance on a variety of tasks, including language modeling, machine translation, and text production, has been attained using transformer-based models in the context of natural language processing (NLP).

4.5.4. Brainstorming

Table 14: Brainstorming Findings

Criteria	**Finding**
Transformer-based neural network model for source code generation.	BERT, GPT-2, and T5 are a few instances of well-known transformer-based models utilized in NLP. These models can be refined for certain further applications like natural language voice commands or source code production because they are pre-trained on vast corpus of text data.
Converting natural language commands to source code	For developing particular patterns and rules for turning natural language input into code, a rule-based approach can be helpful. When the input is clear and consistent, such as when turning spoken programming language commands into source code, this can be helpful.
Voice to source code model process	To successfully convert speech input to source code, the voice input must first be processed. Commonly utilized methods include feature extraction, standardization, and data augmentation. Best suitable voice recognition library can be used for voice to text generation.

4.5.5. Summary of Findings

Table 15: Summary of Findings

Findings	**Literature Review**	**Interviews**	**Observation**	**Brainstorming**
Voice to source code generating tool for quorum programming language will provide an accessible learning environment for the differently abled students.		✓	✓	
Programming with complex syntax consumes more time and voice strains for students.		✓	✓	
T5-small is more suitable language model for natural language commands to source code generation for limited dataset.	✓	✓	✓	
More time consuming for the differently abled students than the normal students to code by dictating each word of the syntax relevant to the programming language by voice.	✓	✓	✓	
Typing is a physical act, making it challenging for students with upper limb impairments to use these programming tools with traditional mouse and keyboard usage.	✓		✓	
Transformer-based neural network model for source code generation.			✓	✓
Determining the most effective language model for text-to-source-code creation which would satisfy the study's complete requirements and offer a way to build voice to source code generation model.	✓		✓	✓

4.6. Context Diagram

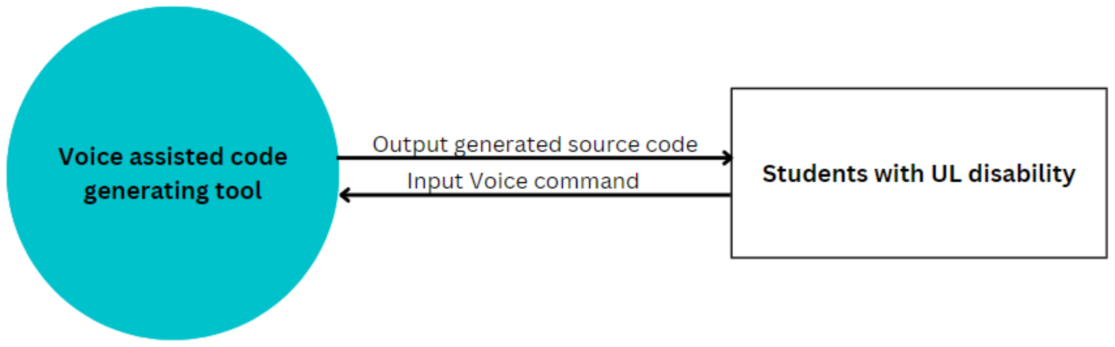

Voice assisted code generating tool
Output generated source code
Input Voice command
Students with UL disability

Figure 6: Context Diagram

4.7. Use case diagram

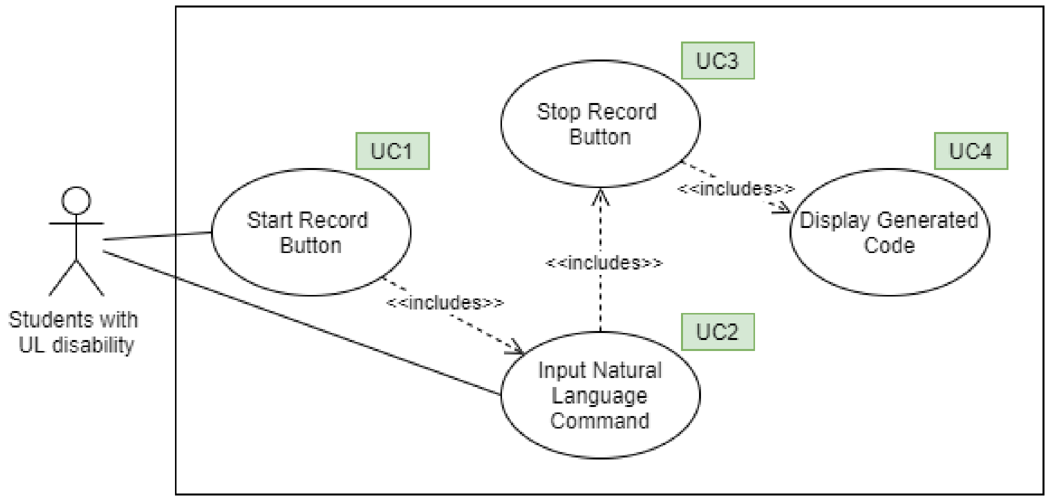


Figure 7: Use Case Diagram-

4.8. Use case descriptions

Table 16: Use Case 01

Use Case 1	Start Record Button
Id	UC:01
Description	Enable the start record button to input natural language command
Primary Actor	Students with UL disability
Supporting Actors	None
Pre–Conditions	None
Post–Conditions	Generated relevant source code snippet.

Trigger	Enable the start record button	
Main Flow	1	User access the tool in website
	2	User inputs natural language voice command
	3	Relevant source code will get displayed
Alternative Flows	No voice command given after enabling the start button	

Table 17: Use Case 02

Use Case 2	Input natural language command	
Id	UC:02	
Description	Input necessary natural language voice command	
Primary Actor	Students with UL disability	
Supporting Actors	None	
Pre–Conditions	Enable the start record button to input natural language command	
Post–Conditions	Generated relevant source code snippet.	
Trigger	Enable the start record button	
Main Flow	1	User enables the start button
	2	User inputs natural language voice command
	3	Relevant source code will get displayed
Alternative Flows	Invalid voice command given.	

4.9. Requirements

The functional and non-functional requirements of the proposed tool are described in detail in this section. The requirements are displayed in order of priority using the MoSCoW prioritization technique.

Table 18: MoSCoW Table

Symbol	**Level**	**Description**
M	Must have	Essential features that must be included in the system
S	Should have	Features that are important to the system that are necessary but not required.

C	Could have	Optional requirements that, depending on time constraints, might be implemented.
W	Will not have	Requirements that are out of scope

4.9.1. Functional Requirements

Table 19:Functional Requirements

ID	**Requirement**	**Priority Level**	**Use Case**
FR1	Students must be able to generate print statement.	M	UC2, UC4
FR2	Students must be able to generate if code block.	M	UC2, UC4
FR3	Students must be able to generate if else code block.	M	UC2, UC4
FR4	Students must be able to generate while loop code block.	M	UC2, UC4
FR5	Students must be able to generate while loop code block with variation.	M	UC2, UC4
FR6	Students must be able to generate basic function code block.	M	UC2, UC4
FR7	Students must be able to generate basic function code block with variation.	M	UC2, UC4
FR8	Students must be able to insert variables.	M	UC2, UC4
FR9	Students must be able to insert variables and assign values.	M	UC2, UC4
FR10	Students must be able to generate basic class structure.	M	UC2, UC4
FR11	Students should be able to generate if else code block with variation.	S	UC2, UC4
FR12	Students should be able to generate if block with variation.	S	UC2, UC4
FR13	Students could be able to generate basic class structure with variation.	C	UC2, UC4
FR14	Students could be able to generate complex class structure with variation	C	UC2, UC4
FR15	Students could be able to generate complex actions code structure with variation	C	UC2, UC4

4.9.2. Non-Functional Requirements

Table 20: Non Functional Requirements

ID	Requirement	Description
NFR1	Performance	The tool should be able to process enormous amounts of information rapidly and effectively, and it should have low latency.
NFR2	Usability	The tool should be compatible with the selected platform, have an intuitive user interface, and be simple to use.
NFR3	Reliability	The tool should be efficient and dependable, capable of managing unforeseen inputs.
NFR4	Scalability	The tool must be able to handle a large user base and scale up as necessary to satisfy escalating demand.
NFR5	Accessibility	A wide spectrum of users, including those with upper limb disabled or restricted with mobility, should be able to use the product.
NFR6	Adaptability	The tool should be versatile and flexible, able to take into account changes in the development environment or programming language.

4.10. Chapter Summary

This chapter described the various elicitation approaches used to determine the requirements. To define and improve the requirements for the research, techniques like literature review, interviews, observations and brainstorming were used. Voice strain in dictating strict syntaxes, code generation in program code through natural language voice commands were found as highly crucial requirements from the requirements gathered from the possible stakeholders. Functional and non-functional requirements were also defined from the requirements gathered.

CHAPTER 5: SOCIAL, LEGAL, ETHICAL AND PROFESSIONAL ISSUES (SLEP)

5.1. Chapter Overview

This section will provide an overview of the chapter, its objectives, and the SLEP issues that was faced during researching and developing the natural language voice commands to source code generation tool.

5.2. SLEP issues and mitigation

5.2.1. Social Issues

- Interview replies were anonymously incorporated to the thesis in a way that would reveal the respondents' individual viewpoints. There was only a summary of the interviewees' remarks recorded.
- Ensured that the viewpoints of the respondents were accurately represented, a summary of their remarks was recorded, and this summary provides a fair and balanced representation of their opinions to avoid any personal bias or interpretation.

5.2.2. Legal Issues

- Issues related to the ownership of the generated code, especially if it is used in commercial applications was considered.
- The usage of all programming languages, tools, and frameworks was authorized by an open source license.
- The privacy of the user's data was maintained, and sensitive information wasn’t included in any part of the research.

5.2.3. Ethical Issues

- Proper information and explanation was given about the project and the purpose of developing the tool to stakeholders, including team members, clients, investors, and end-users, to have a clear understanding of the project and its purpose.
- Prior consent was taken to from the participants before collecting responses since personal medical information was involved for the collected data. Consent was given that the participants were informed about the research purpose, the data collection procedures, the

potential risks, and benefits of participating, and how the data will be used and protected for the research.

5.2.4. Professional Issues

- The entire study procedure was conducted in accordance with very high standards.
- The project's results are accurate and unedited documented in accordance with their natural course.
- Unbiased dataset was created to train and generate the source code

5.3. Chapter Summary

The chapter on social, legal, ethical, and professional issues (SLEP) in the development of natural language voice commands to source code generation tools highlights the potential concerns and implications that were raised, and the mitigation strategies used for it. Overall, the chapter emphasizes the importance of considering these issues in the development and implementation of natural language voice commands to source code generation tools to ensure that the technology is developed and used in an ethical and responsible manner.

CHAPTER 6: DESIGN

6.1. Chapter overview

This chapter includes the selected architecture for the implementation and the discussion of the tiers of the proposed architecture. High-level design, low-level design, and UI wireframes have all been provided to this chapter to clarify how the design goals are expected to be achieved and to provide explanation for particular design choices.

6.2. Design Goals

Table 21: Design Goals

Design Goal	Description
Accuracy	The quality of the underlying NLP and machine learning algorithms, as well as the complexity and variety of the programming languages and coding styles that this tool supports, will all have an impact on how accurate a voice to source code generation. It is decided to use a combination of machine learning and natural language processing (NLP) approaches to evaluate spoken language and produce code that accurately reflects the user's intents in order to increase accuracy.
Efficiency	The complexity of the input instructions, the length and diversity of the codebase, and the technology and processing resources accessible will all have an impact on how effective is a voice to source code generation. It is decided to use efficient algorithms and data structures that can quickly and accurately process massive amounts of natural language input in order to increase efficiency.
Usability	The user's demands is taken into consideration while designing the interface, and it offers straightforward instructions and feedback to help the user through the code generation process. Natural language processing (NLP) and machine learning techniques is decided to use to enhance usability by allowing a tool to identify and adjust to a user's speech patterns and vocabulary.

Integration	The tool will include plugins or extensions that make it possible for it to function in the selected programming environment in order to increase integration. These plugin is planned to keep it simple to install and set up, and they will offer a simplified, simple-to-use interface that is appropriate for the target environment.
Accessibility	The ability of a voice to source code generating extension tool to be used by a variety of users is referred to as its accessibility. While designing the solution the differently abled people are mainly considered to provide an accessible learning environment for them.

6.3. High level Design

6.3.1. Architecture Diagram

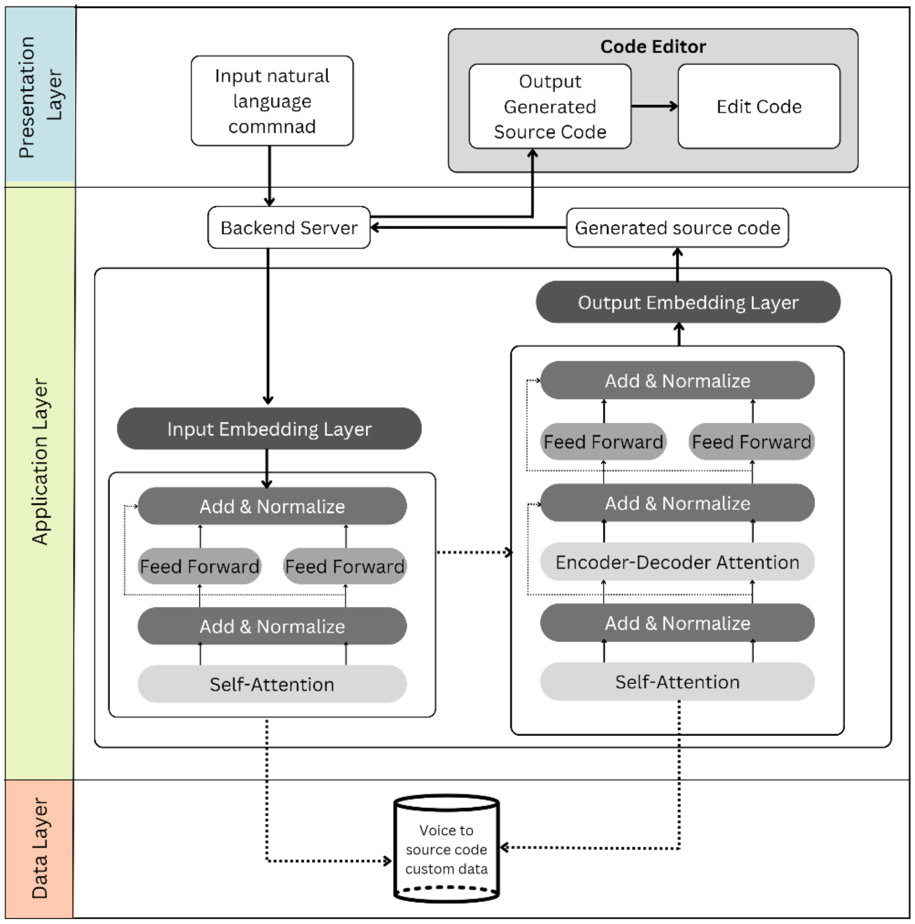


Figure 8: Architecture Diagram

6.3.2. Discussion of layers of the Architecture

Table 22: Discussion of layers of the Architecture

Architecture Layers	Components	Discussion
Presentation Layer	Input natural language command	The user can input start the record button and provide the relevant voice command to generate the source code snippet.
	Code Editor	The generated code will be displayed in the code editor. The user can edit the code with the editor as well.
Application Layer	Backend Server	The backend server is the server that receives HTTP requests from the user's web browser and actually processes the request and returns the response to back to the user's browser.
	Input Embedding Layer	This layer converts the input text into a sequence of vectors that represent the meaning of the text. Each vector represents a token in the input text and is learned through a combination of character-level and word-level embeddings.
	Encoder Layers	The encoder is composed of a stack of identical transformer blocks that transform the input sequence of vectors into a sequence of encoded vectors. Each transformer block consists of multi-head attention and feedforward neural network layers.
	Decoder Layers	The decoder is also composed of a stack of identical transformer blocks that take the encoded input sequence and generate the output sequence. The decoder also uses multi-head attention and feedforward neural network layers, but also includes an additional cross-attention mechanism that helps the decoder attend to the input sequence

	Output Embedding Layer	This layer converts the output sequence of vectors into a sequence of tokens that represent the final output text. Each token is predicted based on the previous tokens in the sequence and is learned through a combination of character-level and word-level embeddings.
Data Layer	Voice to source code data	The dataset used to train and test the voice to source code generation model.

6.4. Low-level Design/ System Design

3.4.1 Choice of design paradigm

SSADM is suitable provides a clear and structured approach to software development and each phase of the development process must be completed before moving on to the next phase. The SSADM methodology was chosen for analysis and development purposes of this research since the primary area of research tends toward data science and it provides a structured approach and reusability of modules. The time frame that must be followed for implementing and documenting this research was also a reason for the selection of this approach.

6.5. Design Diagrams

6.5.1. Component Diagram

The identified components and their data flow are shown in the figures below.

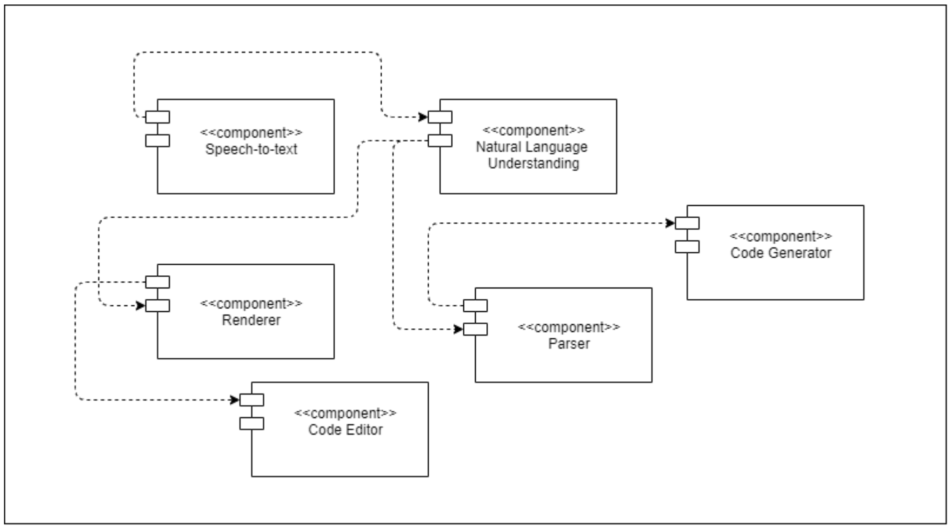


Figure 9: Component Diagram

6.5.2. Data Flow Diagram

Level 1 – DFD

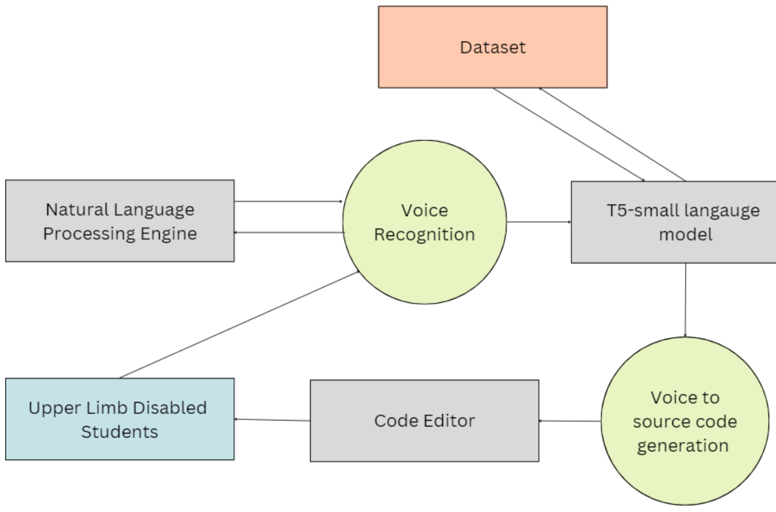

Dataset
Natural Language Processing Engine
Voice Recognition
T5-small langauge model
Upper Limb Disabled Students
Code Editor
Voice to source code generation

Figure 10: Level 1 - DFD

Level 2 – DFD

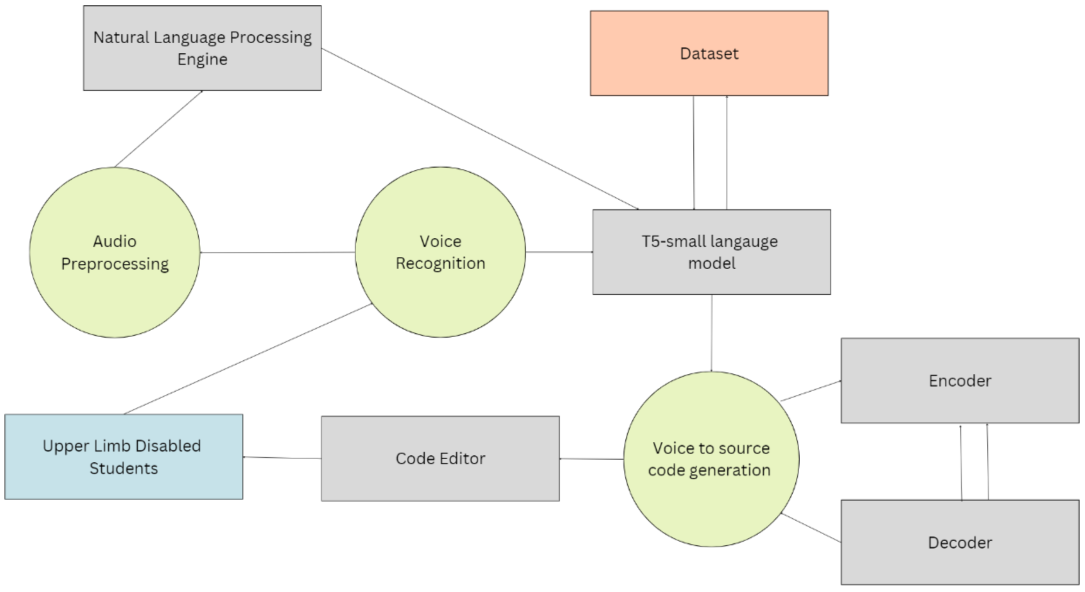

Natural Language Processing Engine
Dataset
Audio Preprocessing
Voice Recognition
T5-small langauge model
Encoder
Upper Limb Disabled Students
Code Editor
Voice to source code generation
Decoder

Figure 11: Level 2 - DFD

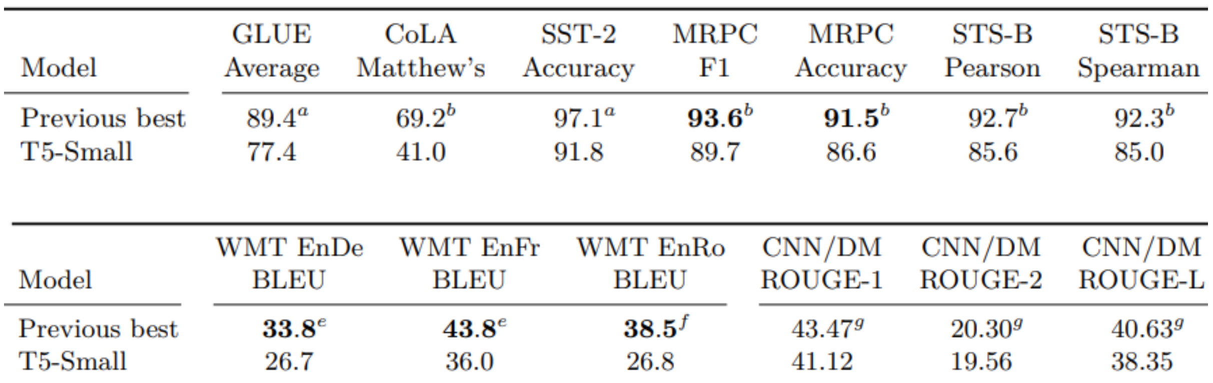
6.5.3. Algorithmic Analysis

The T5-Small transformer-based language model is based on the transformer architecture, which is a type of neural network architecture that uses self-attention to process input data. This allows the model to process variable-length sequences of data, making it well-suited for tasks such as natural language processing.

The T5-Small model is trained using a combination of unsupervised and supervised learning techniques. It is trained on a large corpus of text data using unsupervised pre-training, which involves predicting missing words or sentences in a text corpus. This pre-training is then followed by supervised fine-tuning on specific natural language processing tasks such as question answering or text summarization.

The architecture of the T5-Small model consists of 6 layers of transformer blocks, each with 256 hidden units and 4 attention heads. The model has a total of 60 million parameters, making it smaller and more efficient than larger T5 models. The T5-Small model uses a sequence-to-sequence (Seq2Seq) architecture, which is commonly used in natural language processing tasks. The model consists of an encoder that processes the input data and a decoder that generates the output data. The encoder uses self-attention to process the input sequence and generate a set of context vectors. These context vectors are then used by the decoder to generate the output sequence.

In terms of computational efficiency, the T5-Small model is designed to be highly parallelizable, allowing it to be trained on large-scale distributed systems. The model is also optimized for fast inference, making it suitable for deployment in real-time applications. Evaluated scores of accuracy, precision, recall and F1 Score are given below regarding the performance as a language model. Please refer to Appendix C1 – Algorithmic Analysis for other evaluation scores.



Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0

Model	WMT EnDe BLEU	WMT EnFr BLEU	WMT EnRo BLEU	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-L
Previous best	**33.8**[e]	**43.8**[e]	**38.5**[f]	43.47[g]	20.30[g]	40.63[g]
T5-Small	26.7	36.0	26.8	41.12	19.56	38.35

Figure 12: Algorithm analysis

Perplexity is a measure of how well the model is able to predict the probability distribution of the next word in a sequence. It is calculated as the inverse of the geometric mean of the probability assigned to each word in the test set:

$$Perplexity = exp\left(- \frac{1}{N} \sum_{i=1}^{N} \log p(w_i) \right)$$

Figure 13: Perplexity Equation

where N is the number of words in the test set, w_i is the i-th word, and $p(w_i)$ is the probability assigned to the word by the model.

BLEU (Bilingual Evaluation Understudy) is a measure of the similarity between the generated and reference (human-written) sentences. It ranges from 0 to 1, with 1 indicating perfect similarity. BLEU is calculated as:

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \log p_n \right)$$

Figure 14: BLEU Equation

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is another measure of the similarity between the generated and reference sentences, but it places more emphasis on recall (the number of overlapping words between the generated and reference sentences) than precision (the number of overlapping words divided by the number of words in the generated sentence). ROUGE is typically calculated for different n-gram lengths and using different variants such as ROUGE-1, ROUGE-2, and ROUGE-L.

6.5.4. User Interface Design

6.5.4.1. Low level fidelity wireframe diagram

Please refer to Appendix C2 - Design

6.5.4.2. High level fidelity diagram

Please refer to Appendix C2 - Design

6.5.5. System Process Flow Chart

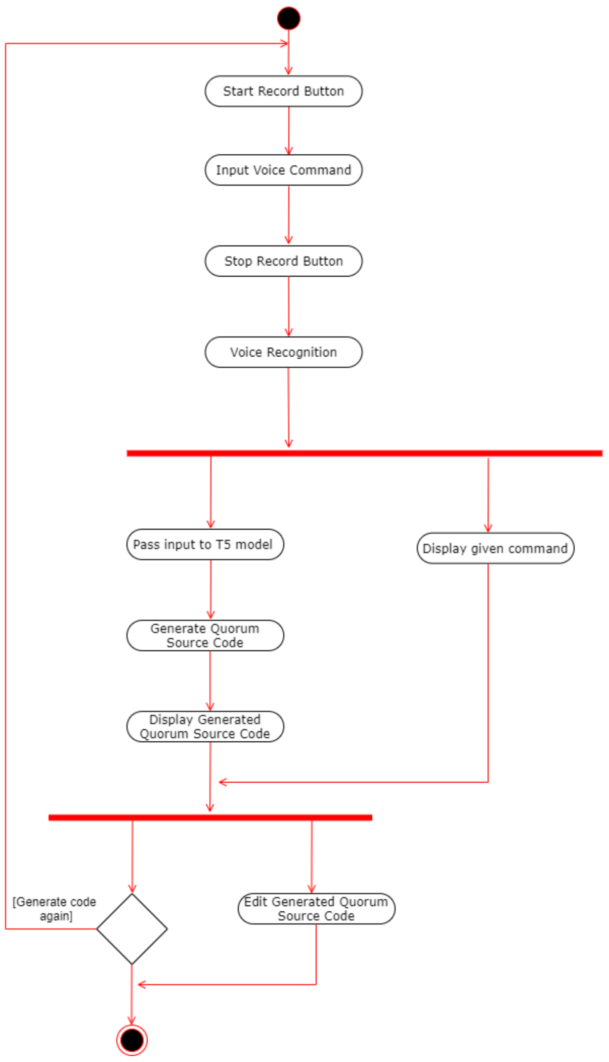


Figure 15: Activity Diagarm

6.6. Chapter Summary

This chapter includes documentation of the project's architecture, architectural features, and system process flow, as well as novel author-designed algorithms. Design goal and system designs are explained in detail with justifications and relevant diagrams. Since SSADM was selected the suitable diagrams for this paradigm are included accordingly. Finally, the planned UI wireframes used to implement the final implementation is included in this chapter.

CHAPTER 7: IMPLEMENTATION

7.1. Chapter Overview

This chapter discusses the implementation of the voice assisted code generation tool as well as the techniques, languages, and supporting technologies that were used in its creation, along with the reasoning for each choice. Based on the specified design in the previous chapter final implementation development process will be explained with code snippets.

7.2. Technology Selection

7.1.1. Technology Stack

The technologies utilized to implement the tool at each tier are displayed below.

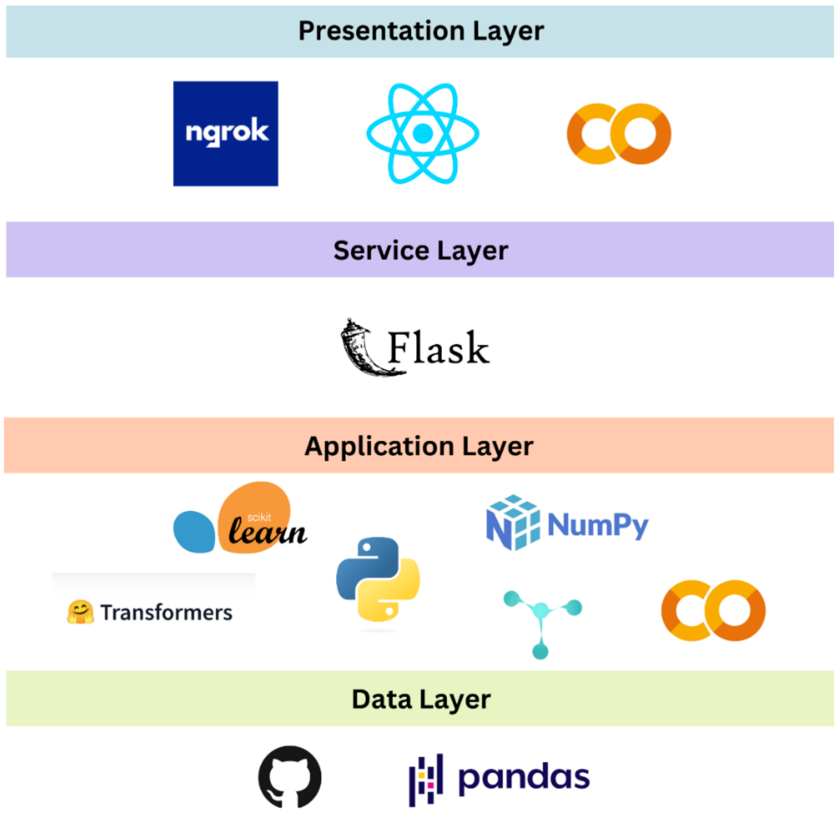


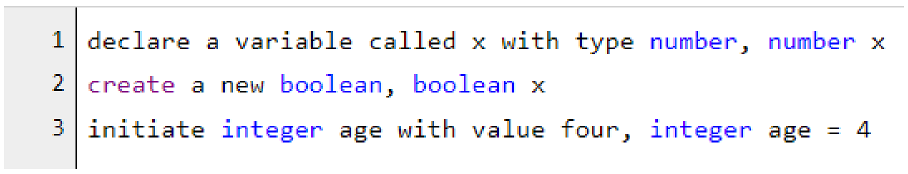
Figure 16: Technology Stack

7.1.2. Data-set Selection

Selecting an appropriate dataset for fine-tuning the T5-small model to generate Quorum language source code from natural language commands is essential for the success of the model. The size of the dataset is critical as it influences the model's ability to learn and generalize well to new inputs. The dataset should be large enough to capture a wide range of natural language commands and their corresponding Quorum language source code variations. The dataset should be diverse enough to cover a wide range of use cases and scenarios. This ensures that the model can generalize well to new input variations. The dataset should be of high quality, with accurate and reliable annotations. This ensures that the model learns the correct mapping between the natural language commands and their corresponding Quorum language source code variations. The dataset should be balanced, meaning it should contain a roughly equal number of positive and negative examples. This ensures that the model does not become biased towards one class of examples. The dataset should be consistent that the same natural language command should always generate the same Quorum language source code variation. Inconsistencies in the dataset can lead to confusion and poor performance.

The scope of the dataset was defined to specify the tasks and use cases for which the Quorum language code will be generated. Types and variables, operators, output and user input, type casting, commenting code, conditional statements with if, repeat, loops and basic structures of classes and actions of Quorum syntax are included to the scope to generate source code in the custom dataset.

A diverse set of natural language commands was collected according to this specific use cases and scope. The range of inputs that the voice assistant is expected to handle was considered when representing the commands. For each natural language command, defined the expected Quorum language code output that the model should generate and annotated each natural language command with its corresponding Quorum language code output. Example snippet is provided on how the command and code were mapped in the dataset.



```
declare a variable called x with type number, number x
create a new boolean, boolean x
initiate integer age with value four, integer age = 4
```

Figure 17: Dataset snippet

Cleaned and preprocessed the dataset to remove any noise or errors in the data. This involved removing duplicates, correcting spelling errors, or removing irrelevant data. The dataset into training, validation, and testing sets. The training set is used to train the model, while the validation and testing sets are used to evaluate the model's performance. The annotated dataset to fine-tune the T5-small model for generating Quorum language code from natural language commands.

7.2.3. Development Frameworks

Table 23: Development Frameworks

Framework	**Justification**
Flask	Python-based micro web framework for building web applications. It also uses libraries to provide a secure tunnel for testing the web application on a local machine. It can be used for building small to medium-sized web applications, APIs, and even complex systems.
React	Allows to build reusable UI components and compose them together to create complex user interfaces. React uses a declarative approach, to achieve the desired outcome. This makes it easier to reason about and maintain complex UI code.

7.2.4. Programming Languages

For deep learning and natural language processing (NLP) activities, Python is a widely used programming language. It has a wide range of libraries and tools, including speech recognition, natural language processing, and machine learning libraries, that may be used to create voice-to-source code generation models. For this reason, python is selected as the language for the development of the model. It provides numerous tools and libraries that can be used to create natural language processing and machine learning tasks.

JavaScript is the primary programming language used for front-end development with React. JavaScript is also used since the frontend is developed using the React framework for the users to interact and generate the necessary code blocks.

7.2.5. Libraries

Table 24: Libraries selection and Justification

Library	**Justification**
Transformers	A library for natural language processing tasks that provides pre-trained models and tools for fine-tuning them on custom datasets
Numpy	Large, multidimensional arrays and matrices can be supported, and a variety of mathematical operations can be carried out on these arrays using the extensive range of functions provided.
Pandas	It offers quick, adaptable, and expressive data structures that make working with structured data in many different file types—including CSV, Excel, SQL databases, etc.
Torch	A library for deep learning that provides tools for building and training neural networks
Sklearn	A library for machine learning that provides tools for data preprocessing, model selection, and evaluation

7.2.6. IDE

Table 25: IDE Selection and Justification

IDE	**Justification**
Google Colab	A Google provided cloud-based Jupyter notebook environment that allows to run and execute Python code directly from a web browser without the need for installation or setup. It is a widely used platform for deep learning research and instruction since it offers a free GPU and TPU runtime for running demanding machine learning models and algorithms.

7.2.7. Summary of Technology Selection

Table 26: Summary of Technology Selection

Component	**Tools**
Development Framework	Flask
Programming Languages	Python, HTML, CSS, Javascript

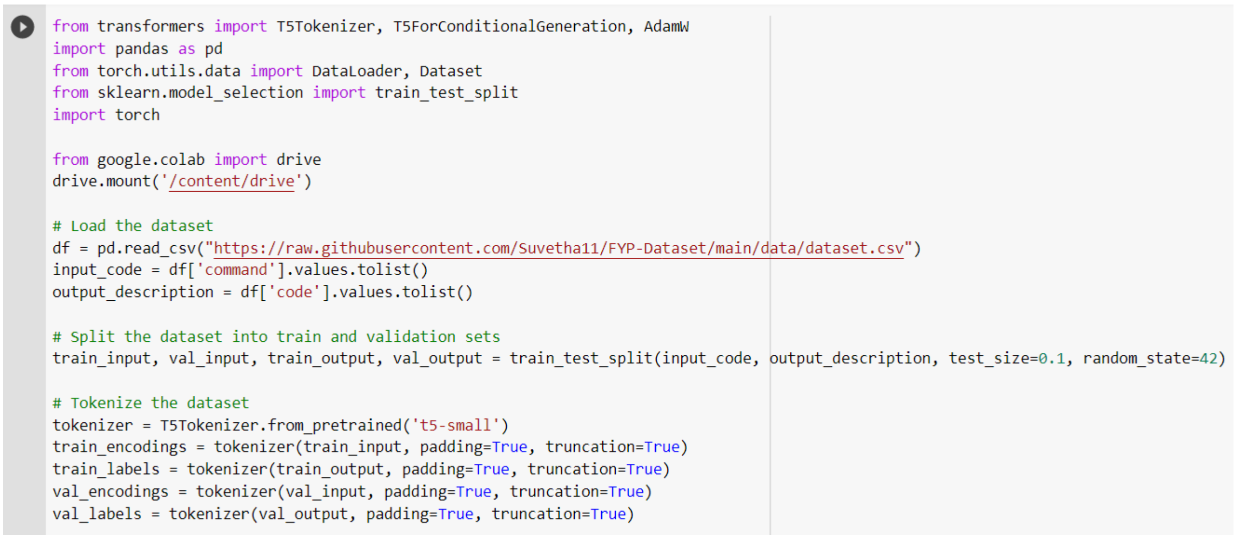
Libraries	Transformers, Numpy, Pandas, Torch, Sklearn
IDE	Google Colab

7.3. Implementation of the Core Functionality

The dataset was created annotating the natural language commands and Quorum syntax as mentioned in the dataset selection section and the custom dataset was used to train the t5-small variant of T5 (Text-to-Text Transfer Transformer) transformer based language model was used for code generation since the custom dataset was limited. Hyperparameters such as the learning rate and batch size may need to be decreased when training on a smaller dataset to prevent overfitting. Hyperparameters such as the number of layers or attention heads may need to be reduced in order to prevent the model from becoming too complex and overfitting on the training data. Comparatively the hyperparameters for t5-small variant was more promising than the other large language models for finetuning with the custom dataset.

Training the language model

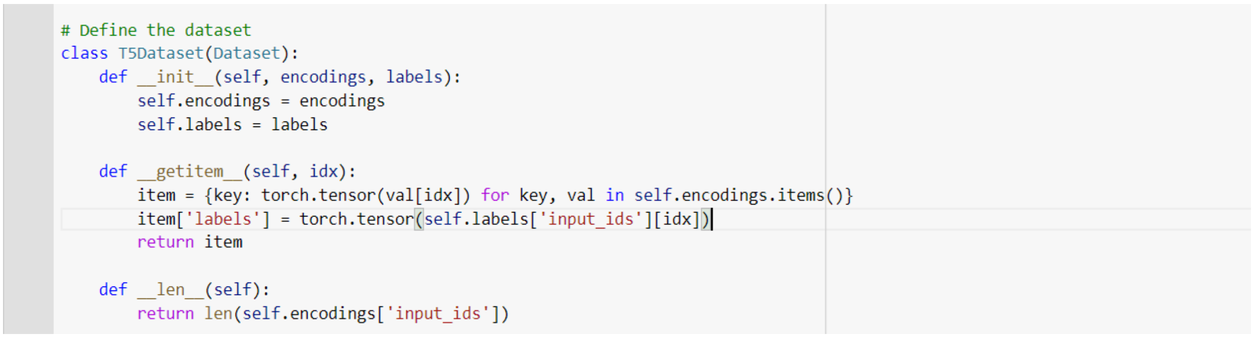
The custom dataset is loaded and split into training and validation sets using the train_test_split function from scikit-learn. Tokenized the input natural language commands and output Quorum language code using the T5Tokenizer from the transformers library, with the pre-trained T5-small model and created PyTorch tensors for the tokenized inputs and outputs.



```
from transformers import T5Tokenizer, T5ForConditionalGeneration, AdamW
import pandas as pd
from torch.utils.data import DataLoader, Dataset
from sklearn.model_selection import train_test_split
import torch

from google.colab import drive
drive.mount('/content/drive')

# Load the dataset
df = pd.read_csv("https://raw.githubusercontent.com/Suvetha11/FYP-Dataset/main/data/dataset.csv")
input_code = df['command'].values.tolist()
output_description = df['code'].values.tolist()

# Split the dataset into train and validation sets
train_input, val_input, train_output, val_output = train_test_split(input_code, output_description, test_size=0.1, random_state=42)

# Tokenize the dataset
tokenizer = T5Tokenizer.from_pretrained('t5-small')
train_encodings = tokenizer(train_input, padding=True, truncation=True)
train_labels = tokenizer(train_output, padding=True, truncation=True)
val_encodings = tokenizer(val_input, padding=True, truncation=True)
val_labels = tokenizer(val_output, padding=True, truncation=True)
```

Figure 18: Model Training

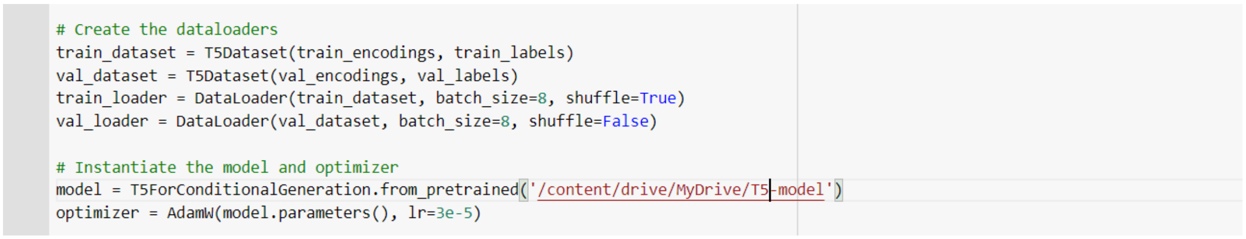
Defined a custom PyTorch Dataset class called T5Dataset for the T5 model. The class takes in encodings and labels as inputs, which are the tokenized input and output sequences obtained using the T5Tokenizer. The __getitem__ method returns a dictionary with keys as the tokenization output keys and corresponding tensor values. The labels key contains the input sequence tokenized using the tokenizer. The __len__ method returns the length of the input sequences. This custom dataset class is required to use the DataLoader class of PyTorch for efficient loading and batching of data during training.



```
# Define the dataset
class T5Dataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels['input_ids'][idx])
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])
```

Figure 19: Defining dataset

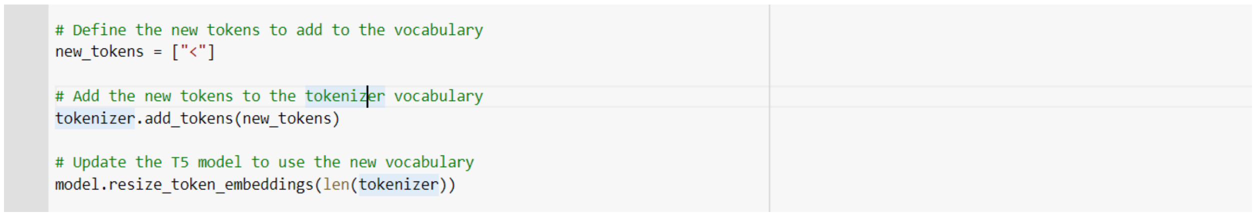
The dataloaders were created for the training and validation datasets using the T5Dataset class defined earlier. train_loader and val_loader are instances of DataLoader that allow for iterating over the dataset in batches of 8 during training and validation. The T5 model is then instantiated accessing the model from google drive, which loads a pre-trained version of the T5 model stored in the specified directory. The optimizer used for training the model is AdamW with a learning rate of 3e-5.



```
# Create the dataloaders
train_dataset = T5Dataset(train_encodings, train_labels)
val_dataset = T5Dataset(val_encodings, val_labels)
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)

# Instantiate the model and optimizer
model = T5ForConditionalGeneration.from_pretrained('/content/drive/MyDrive/T5-model')
optimizer = AdamW(model.parameters(), lr=3e-5)
```

Figure 20: Create data loaders

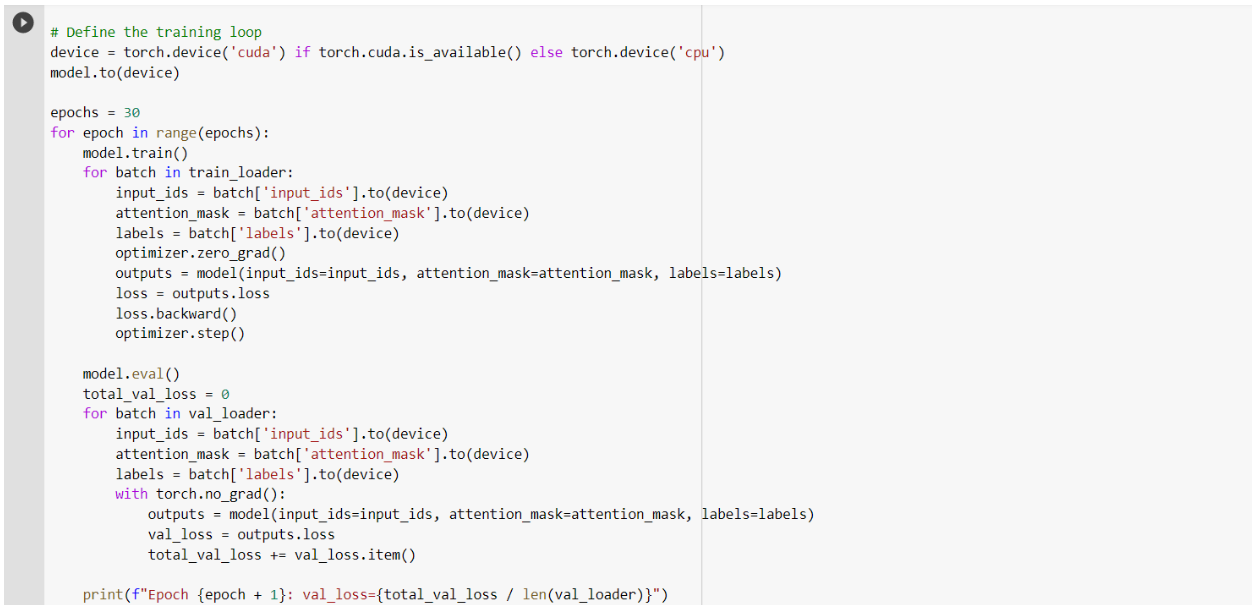
Added a new token "<" to the existing T5 tokenizer vocabulary and resizes the token embeddings of the T5 model to reflect the updated vocabulary size. This is necessary because the added token needs to be included in the model's vocabulary to be properly recognized and used during training and inference.



```
# Define the new tokens to add to the vocabulary
new_tokens = ["<"]

# Add the new tokens to the tokenizer vocabulary
tokenizer.add_tokens(new_tokens)

# Update the T5 model to use the new vocabulary
model.resize_token_embeddings(len(tokenizer))
```

Figure 21: Add new token

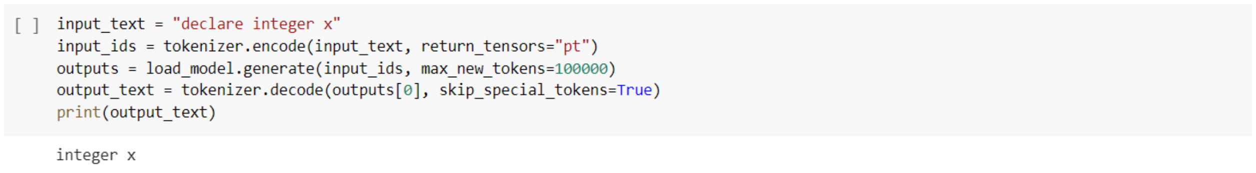
Moved the input_ids, attention_mask, and labels tensors to the appropriate device (GPU or CPU). Reset the gradients of the optimizer. Passed the input_ids and attention_mask tensors to the model, along with the labels for teacher forcing. Computed the loss based on the output of the model. Performed backpropagation to compute the gradients and update the model parameters. After each epoch of training, the loop evaluates the model's performance on the validation dataset. At the end of each epoch, the average validation loss for that epoch was printed to monitor how well the model is generalizing to unseen data.



```
# Define the training loop
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)

epochs = 30
for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        optimizer.zero_grad()
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

    model.eval()
    total_val_loss = 0
    for batch in val_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        with torch.no_grad():
            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
            val_loss = outputs.loss
            total_val_loss += val_loss.item()

    print(f"Epoch {epoch + 1}: val_loss={total_val_loss / len(val_loader)}")
```

Figure 22: Trianing T5-small with custom dataset

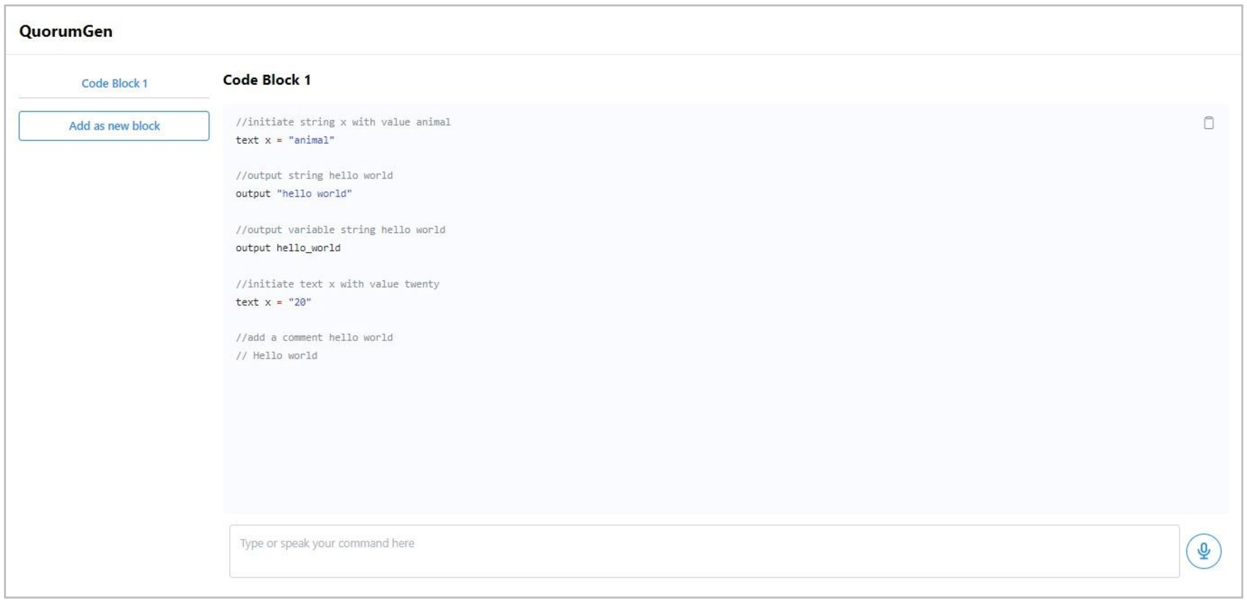
Fine-tuned T5 model was tested whether it generates the specific code for the given natural language input.



```
[ ] input_text = "declare integer x"
    input_ids = tokenizer.encode(input_text, return_tensors="pt")
    outputs = load_model.generate(input_ids, max_new_tokens=100000)
    output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    print(output_text)

    integer x
```

Figure 23: Testing with input command

7.4. User Interface



```
//initiate string x with value animal
text x = "animal"

//output string hello world
output "hello world"

//output variable string hello world
output hello_world

//initiate text x with value twenty
text x = "20"

//add a comment hello world
// Hello world
```

Figure 24: User Interface

7.5. Chapter Summary

The chapter included the technologies, languages, and tools used to develop the final implementation that was created for the research. Flask, Transformers, Numpy, Pandas, Torch, Sklearn and Google Colab are the technologies handled for this final implementation and further implementation is expected to be done in future. Code snippets and model developed as the core functionality are accompanied by discussions in the core functionality section with the complete implementation.

CHAPTER 8: TESTING

8.1. Chapter Overview

The testing done at various levels to make sure that functions flowed as expected is covered in this chapter. The idea of software testing, its goals and objectives, and the several kinds of testing that were carried out on the created model and tool are all covered in this section. This chapter also discusses various testing methodologies and obtained outcomes.

8.2. Objectives and Goals of Testing

Testing is a crucial part of software development as it helps to identify errors, defects, and bugs in software applications before they are released to end-users. The primary objectives and goals of testing are:

- Finding defects: The primary objective of testing is to find defects in the software application. By detecting defects early, developers can fix them before the product is released to the market.
- Ensuring quality: Testing helps to ensure the quality of the software application. It ensures that the application is free from defects, meets user requirements, and functions as expected.
- Enhancing reliability: Testing helps to enhance the reliability of the software application by detecting and fixing defects, thus improving the stability and consistency of the application.
- Reducing costs: By detecting and fixing defects early in the development process, testing helps to reduce the costs associated with fixing defects in later stages of development or after the product has been released.
- Meeting user requirements: Testing helps to ensure that the software application meets user requirements and functions as expected.
- Increasing customer satisfaction: By delivering a high-quality product that meets user requirements and functions as expected, testing helps to increase customer satisfaction and build a positive reputation for the organization.
- Compliance with industry standards: Testing helps to ensure that the software application complies with industry standards and regulations, such as security and privacy standards.

8.3. Testing Criteria

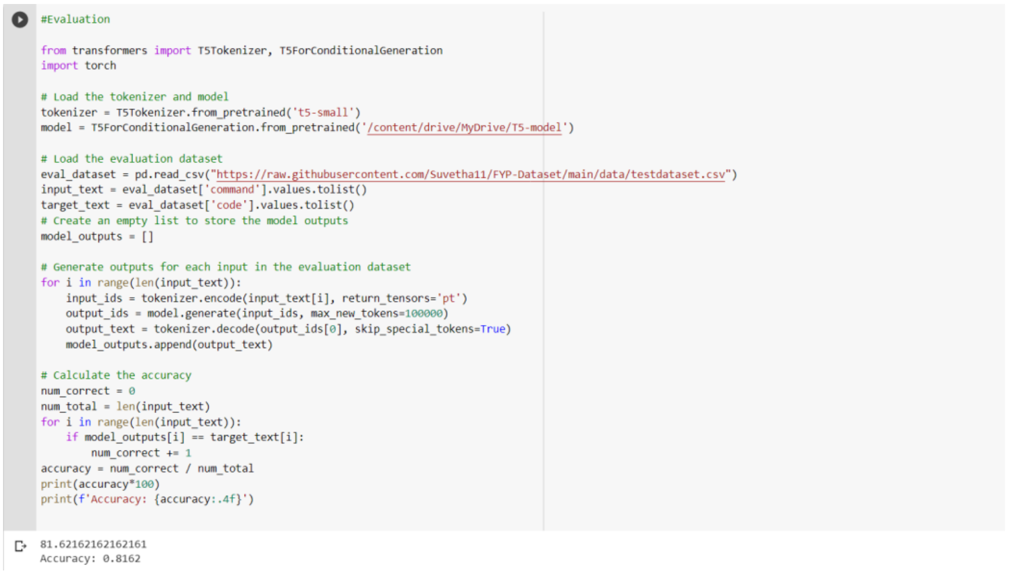
Testing criteria refer to the attributes or characteristics that a software system should meet to be considered functional, reliable, efficient, maintainable, and usable. Testing criteria are essential because they help define the metrics that the software system needs to meet to be considered acceptable.

The testing is based on the next two factors.

1. Functional Quality- Used to determine whether functions are functioning as intended.
2. Structural Quality - This gauges how well a system performs a specific task.

8.4. Model Testing & Evaluation

The evaluation involves generating Quorum language code for each input command using the T5 model and comparing it with the corresponding human-generated code. The evaluation metric used is accuracy, which is calculated by counting the number of correctly generated code samples and dividing it by the total number of samples in the dataset. After generating the code for each input, the accuracy of the model was calculated by comparing the generated code with the human-generated code in the dataset. Correctly generated code samples count is divided by the total number of samples in the dataset to get the accuracy. The model accuracy to generate code was 81.62% including new variations in the testing dataset.

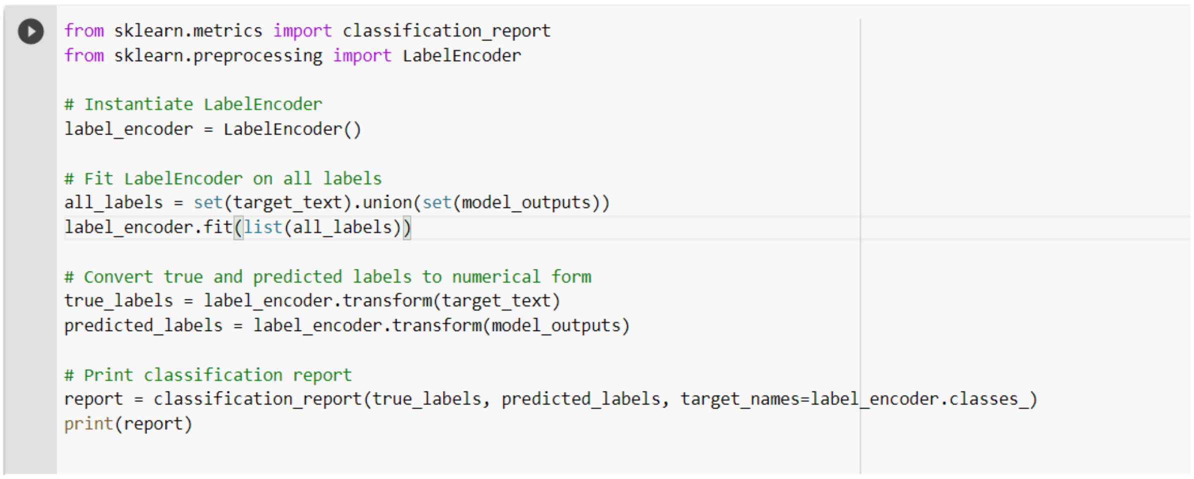


```python
#Evaluation

from transformers import T5Tokenizer, T5ForConditionalGeneration
import torch

# Load the tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('t5-small')
model = T5ForConditionalGeneration.from_pretrained('/content/drive/MyDrive/T5-model')

# Load the evaluation dataset
eval_dataset = pd.read_csv("https://raw.githubusercontent.com/Suvetha11/FYP-Dataset/main/data/testdataset.csv")
input_text = eval_dataset['command'].values.tolist()
target_text = eval_dataset['code'].values.tolist()
# Create an empty list to store the model outputs
model_outputs = []

# Generate outputs for each input in the evaluation dataset
for i in range(len(input_text)):
    input_ids = tokenizer.encode(input_text[i], return_tensors='pt')
    output_ids = model.generate(input_ids, max_new_tokens=100000)
    output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    model_outputs.append(output_text)

# Calculate the accuracy
num_correct = 0
num_total = len(input_text)
for i in range(len(input_text)):
    if model_outputs[i] == target_text[i]:
        num_correct += 1
accuracy = num_correct / num_total
print(accuracy*100)
print(f'Accuracy: {accuracy:.4f}')
```

```
81.62162162162161
Accuracy: 0.8162
```

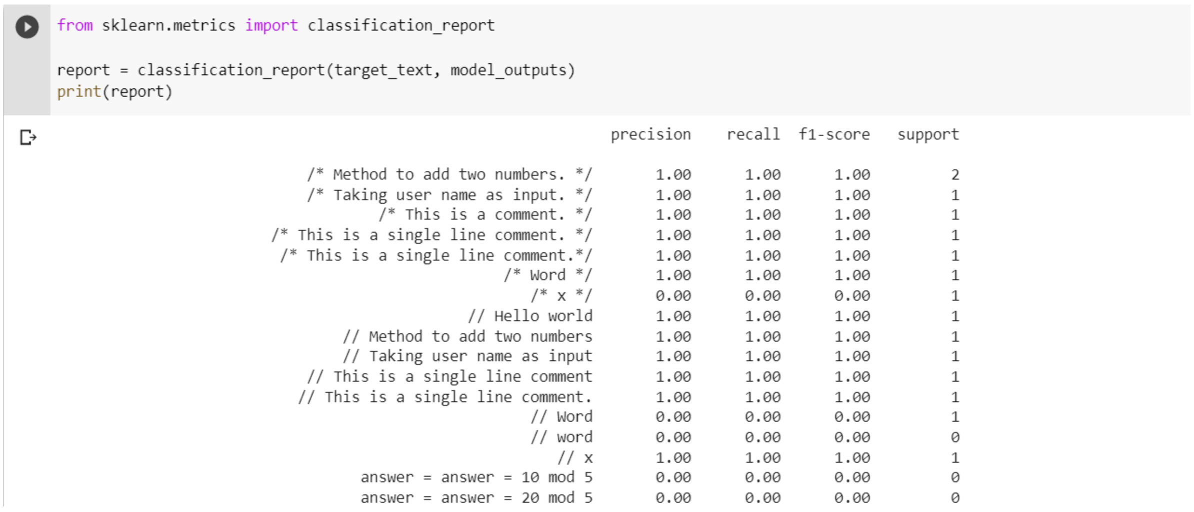
Figure 25: Evaluation

A classification report is a performance evaluation report for a machine learning classification model. It summarizes various metrics such as precision, recall, F1-score, and support for each class in the dataset. It is a useful tool for evaluating the performance of a classification model, especially in multi-class classification problems where there are more than two classes. The report provides an overview of how well the model is performing for each class and can help identify areas where the model needs improvement. Classification report for the evaluation of the T5-small model was generated using sklearn library.



```
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder

# Instantiate LabelEncoder
label_encoder = LabelEncoder()

# Fit LabelEncoder on all labels
all_labels = set(target_text).union(set(model_outputs))
label_encoder.fit(list(all_labels))

# Convert true and predicted labels to numerical form
true_labels = label_encoder.transform(target_text)
predicted_labels = label_encoder.transform(model_outputs)

# Print classification report
report = classification_report(true_labels, predicted_labels, target_names=label_encoder.classes_)
print(report)
```

Figure 26: Classification report code

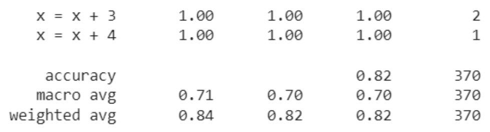


```
from sklearn.metrics import classification_report

report = classification_report(target_text, model_outputs)
print(report)
```

```
                                         precision    recall  f1-score   support

          /* Method to add two numbers. */       1.00      1.00      1.00         2
            /* Taking user name as input. */       1.00      1.00      1.00         1
                    /* This is a comment. */       1.00      1.00      1.00         1
      /* This is a single line comment. */       1.00      1.00      1.00         1
       /* This is a single line comment.*/       1.00      1.00      1.00         1
                                /* Word */       1.00      1.00      1.00         1
                                   /* x */       0.00      0.00      0.00         1
                            // Hello world       1.00      1.00      1.00         1
              // Method to add two numbers       1.00      1.00      1.00         1
              // Taking user name as input       1.00      1.00      1.00         1
          // This is a single line comment       1.00      1.00      1.00         1
         // This is a single line comment.       1.00      1.00      1.00         1
                                   // Word       0.00      0.00      0.00         1
                                   // word       0.00      0.00      0.00         0
                                      // x       1.00      1.00      1.00         1
                  answer = answer = 10 mod 5       0.00      0.00      0.00         0
                  answer = answer = 20 mod 5       0.00      0.00      0.00         0
```

Figure 27: Classification report

Below are the metrics such as precision, recall, f1-score, and support for each class (label). The macro-averaged and micro-averaged metrics was also calculated across all the classes.



```
    x = x + 3       1.00      1.00      1.00         2
    x = x + 4       1.00      1.00      1.00         1

     accuracy                           0.82       370
    macro avg       0.71      0.70      0.70       370
 weighted avg       0.84      0.82      0.82       370
```

Figure 28: Classification report

8.5. Holdout Validation

The custom dataset is split into training and testing sets, and each model is trained on the training set and evaluated on the testing set. The performance metrics of the three models were then compared to determine which one performs better. Please refer to Appendix D - Testing.

Summary of performance metrics of the selected three models.

Table 27:Holdout validation scores

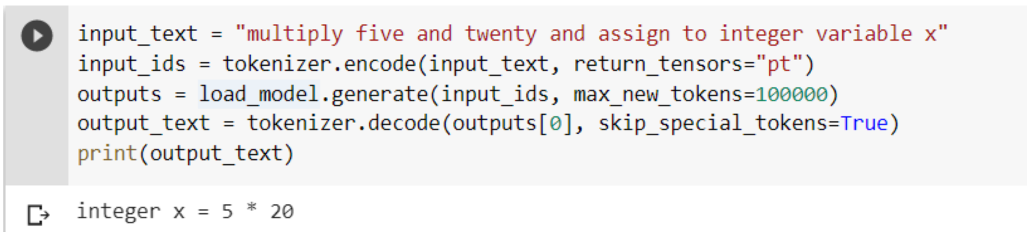
Model	Accuracy	Precision	Recall	F1-score
T5-mini	71.89	0.63	0.63	0.63
T5-small	81.62	0.71	0.70	0.70
T5-base	69.72	0.60	0.60	0.60

T5-small was selected for the implementation of this research according to the evaluation metrics.

8.6. Functional Testing

Testing the model is to check if the trained model generates the specific Quorum code for the given natural language command. The model was tested by giving natural language commands to generate the source code whether it is generating the specific code with correct syntax.

For example, for the given natural language command "multiply five and twenty and assign to integer variable x" the source code was generated as given in the screenshot below.

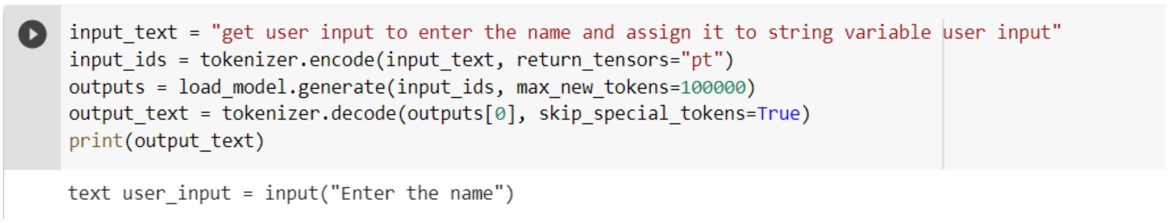


```
input_text = "multiply five and twenty and assign to integer variable x"
input_ids = tokenizer.encode(input_text, return_tensors="pt")
outputs = load_model.generate(input_ids, max_new_tokens=100000)
output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(output_text)
```

```
integer x = 5 * 20
```

Figure 29: Functional Testing 1

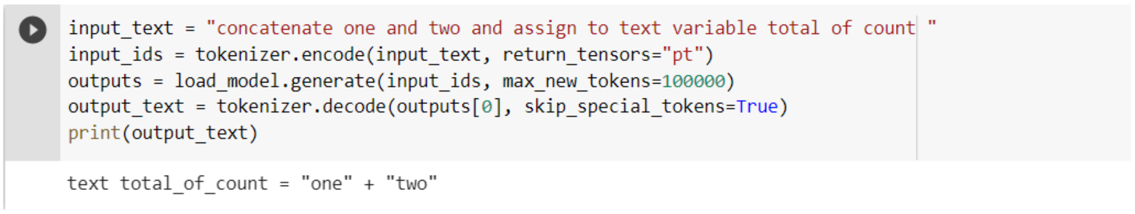
To get the user input code “get user input to enter the name and assign it to string variable user input” command was given and the correct code with specific syntax was generated.



```
input_text = "get user input to enter the name and assign it to string variable user input"
input_ids = tokenizer.encode(input_text, return_tensors="pt")
outputs = load_model.generate(input_ids, max_new_tokens=100000)
output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(output_text)

text user_input = input("Enter the name")
```

Figure 30:Functional Testing 2

Similarly multiple testing was done to test the model to check how far it has learned and in what section the model needs finetuning and accordingly the model was finetuned with validation dataset in the syntax sections identified from the testing carried out.



```
input_text = "concatenate one and two and assign to text variable total of count "
input_ids = tokenizer.encode(input_text, return_tensors="pt")
outputs = load_model.generate(input_ids, max_new_tokens=100000)
output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(output_text)

text total_of_count = "one" + "two"
```

Figure 31: Functional Testing 3

8.7. Module and Integration Testing

Table 28:Model and Integrated Testing

Test ID	Voice Input	Expected Code	Actual Output	Status
ID1	initiate integer y with value eleven	integer y = 11	integer y = 11	Pass
ID2	print variable string hello world	output hello_world	output hello_world	Pass
ID3	subtract nine from seven and assign to integer variable x	integer x = 9 - 7	integer x = 9 - 7	Pass
ID4	concatenate fire and fox and assign to text variable word	text word = "fire" + "fox"	text word = "fire" + "fox"	Pass

ID5	repeat ten times	repeat 10 times	repeat 10 times	Pass

8.8. Non-Functional Testing

Qualitative testing was done for the non-functional requirements mentioned in the SRS chapter by evaluating with experts and users.

8.9. Limitations of the testing process

The model was trained with a limited dataset with limited scope of the syntax. The testing dataset was also limited, and variations were added to an extent due to insufficient dataset. The test result was also affected by the voice recognition library that was used to generate the natural language voice command to text form.

8.10. Chapter Summary

The testing of the model and the implementation is covered in the chapter. A classification report is generated to evaluate the performance of the classification model, and metrics such as precision, recall, f1-score, and support are calculated for each class. The accuracy of the code generation model was 81.62%. Functional testing was carried out to check if the model generates the specific Quorum code for the given natural language command. The model was fine-tuned based on the syntax sections identified from the testing carried out. The limitations of the testing process include a limited dataset and scope of syntax, a limited testing dataset with variations added to an extent, and the voice recognition library used to generate the natural language voice command to text form affecting the test result.

CHAPTER 9: EVALUATION

9.1. Chapter Overview

This chapter discusses how the project was evaluated by different system stakeholders. Their rating is based on their scope and potential for improvement. In order to realize and evaluate the product on a personal level, self-evaluation has been done.

9.2. Evaluation Methodology and Approach

Evaluating a transformer-based language model requires a combination of quantitative and qualitative analysis. Using evaluation techniques for code generation tool that were taken from the literature and applied to the tests conducted in the testing chapter, the research outcome provided by the model was evaluated. This chapter will present the expert feedback that was gathered using thematic analysis.

9.3. Evaluation Criteria

In order to evaluate the various components of the research that required to be evaluated in order to decide the value of the research that was undertaken, the following criteria were utilized for the theme analysis that emerged from expert interviews.

Table 29:Evaluation Criteria

ID	**Criteria**	**Purpose**
EC1	Project Idea	To get feedback on the idea of the voice assisted code generating tool from the stakeholders.
EC2	Scope and depth	To get opinions on the specified scope due to the limited dataset.
EC3	Design, development approach and architecture	To determine whether the system's design and architecture complies with accepted engineering principles and practices.
EC4	Performance and accessibility	To ascertain whether the generated prototype complies with the norms and

		requirements for accessibility and with good performance.
EC5	Possible improvements	To identify improvements that could be worked on as future research associated to the research undertaken.

9.4. Self-Evaluation

Table 30:Self Evaluation

Criteria ID	**Purpose**
EC1	The idea to initiate a voice assisted tool to make learning programming accessible to the differently abled students was the main goal of the research.
EC2	The scope and depth is bounded due to the limited dataset created for the code generation in this research so only narrow scope was covered in the given time constraint.
EC3	The design, development and approach conducted is to the selected problem has valid contribution to both the research and problem domain.
EC4	The performance and accessibility needs more improvement since the current implementation stage model was trained with limited dataset for bounded scope of the Quorum programming syntax. The tool needs complete voice assistance to provide complete access to the tool for the differently abled students.
EC5	The defined scope needs to be extended to cover all sections of syntax of Quorum programming language. Voice commands should be widen in to multilingual commands for code generation to provide more accessibility to start learning programming syntax and concepts without language barriers.

9.5. Selection of the Evaluators

9.5.1. End Users

Table 31:End Users

End User	Purpose
Upper limb disabled students	These individuals are the primary end users of the tool, so it is essential to involve them in the evaluation process to ensure that the tool is accessible, easy to use, and meets their needs.
Caregivers and support staff	These individuals may assist the students in using the tool and can provide feedback on its effectiveness and ease of use.

9.5.2. Experts

Table 32:Experts

End User	Purpose
Assistive Technology Specialists	These individuals have expertise in assistive technology and can provide feedback on the tool's accessibility, compatibility with other assistive technology, and overall usability.
Accessibility Experts	These individuals can assess the tool's compliance with accessibility standards and provide feedback on how to improve the tool's accessibility for individuals with upper limb disabilities.
Software engineers	These individuals can evaluate the tool's code quality, compatibility with existing software systems, and overall technical feasibility.

Natural Language Processing (NLP) experts	These individuals can provide feedback on the tool's language processing capabilities, including accuracy, fluency, and naturalness.
Machine learning/Deep learning experts	These individuals can evaluate the machine learning models used in the tool, including the T5-small model, and provide feedback on its performance and accuracy.

9.6. Evaluation Result

Table 33Evaluation Result

Criteria ID	**Purpose**
EC1	Code generation is a trending research area that can help developers be more productive and produce higher quality code. This initiative to provide access to the differently abled students to learn programming is creditable and needs further improvements on the model and user interface.
EC2	Considering the time constraint and limited dataset, the model and the implemented tool is justified.
EC3	The design considered to implement the project needs enhancement. The voice recognition section can be improved with custom trained model and increase the performance of the tool.
EC4	Since the present implementation stage model was developed with a small dataset for the confined scope of the Quorum programming syntax, efficiency and accessibility still need to be improved. For students with disabilities to have full access to the tool, it must provide total voice help.
EC5	Model needs to be trained in a large dataset with variations for the model to understand the syntax and generate code for new and complex natural language commands. The editor needs complete assistance by voice for accessibility to handle the tool. Voice assistance should be enabled for code edition to make changes to the generated code.

9.7. Limitations of Evaluation

Limitation of the evaluation of the T5-small language model trained with a custom dataset to generate Quorum language code for natural language commands is the lack of direct feedback from upper limb disabled students who could potentially benefit from the tool. Although the evaluation used standard metrics such as accuracy, ROUGE, and BLEU scores to assess the model's performance, it is crucial to obtain feedback from the target users to determine the actual usefulness and effectiveness of the tool in real-world scenarios. The insights and experiences of the users can provide valuable information on the model's strengths, weaknesses, and areas of improvement, which may not be evident from the quantitative evaluation alone. Therefore, future research should consider involving the target users in the evaluation process to obtain a more comprehensive understanding of the model's impact and effectiveness.

9.8. Evaluation on Functional Requirements

Table 34:Evaluation of functional Requirements

ID	**Requirement**	**Priority Level**	**Evaluation**
FR1	Students must be able to generate print statement.	M	Implemented
FR2	Students must be able to generate if code block.	M	Implemented
FR3	Students must be able to generate if else code block.	M	Implemented
FR4	Students must be able to generate while loop code block.	M	Implemented
FR5	Students must be able to generate while loop code block with variation.	M	Implemented
FR6	Students must be able to generate basic function code block.	M	Implemented
FR7	Students must be able to insert variables.	M	Implemented
FR8	Students must be able to insert variables and assign values.	M	Implemented
FR9	Students must be able to generate basic class structure.	M	Implemented

FR10	Students must be able to generate basic class structure with variation.	S	Implemented
FR11	Students should be able to generate if else code block with variation.	S	Implemented
FR12	Students should be able to generate if block with variation.	S	Implemented
FR13	Students could be able to generate basic class structure with variation.	C	Not Implemented
FR14	Students could be able to generate complex class structure with variation	C	Not Implemented
FR15	Students could be able to generate complex actions code structure with variation	C	Not Implemented
Functional Requirement Completion Percentage = 11/14*100 = 78.5%			

9.9. Evaluation on Non-Functional Requirements

Table 35:Evaluation of non-functional Requirements

ID	Requirement	Priority Level	Evaluation
NFR1	Performance	Important	Implemented
NFR2	Usability	Important	Implemented
NFR3	Reliability	Important	Implemented
NFR4	Scalability	Desirable	Not Implemented
NFR5	Accessibility	Desirable	Implemented
NFR6	Adaptability	Desirable	Implemented
Non Functional Requirement Completion Percentage = 5/6*100 = 83.3%			

9.10. Chapter Summary

In this chapter the evaluation methodology and approach used to evaluate the transformer-based language model for code generation presented in the research is discussed. The evaluation process involved a combination of quantitative and qualitative analysis, and the research outcomes were evaluated using evaluation techniques for code generation tools from the literature. The chapter presents the evaluation criteria and self-evaluation results, as well as the selection of evaluators, including end-users and experts. The evaluation results show that the project idea is creditable and requires further improvements, such as enhancing the design and voice recognition section, training the model with a larger dataset, and improving accessibility. However, the evaluation has limitations in terms of direct feedback from target users. Future research should consider involving target users to obtain a comprehensive understanding of the model's impact and effectiveness.

CHAPTER 10: CONCLUSION

10.1. Chapter Overview

This provides a summary of the achievements, challenges, and deviations encountered in the research project aimed at developing a voice-assisted tool for source code generation for students with upper limb disability. It includes a discussion of the research aims and objectives, the learning outcomes, and the utilization of knowledge and skills gained from relevant courses and experiences. The chapter also details the challenges faced during the research and the approaches taken to overcome these challenges. Additionally, the deviations from the original project scope and schedule are discussed, along with the reasons for these deviations.

10.2. Achievements of Research Aims & Objectives

10.2.1. Research Aims

The aim of this research is to design, develop and evaluate a voice assisted tool for source code generation for students with upper limb disability.

The research's aim was effectively met by developing voice assisted source code generating tool for Quorum programming language for natural language voice commands for the upper limb disabled students to learn programming. The system was designed using appropriate research expertise in the field and standard procedures at all stages. Further improvements are expected to be carried out in future.

10.2.2. Research Objectives

Table 36:Achievements of research objectives

Research Objectives	**Learning Outcomes**	**Status**
Problem Identification	LO2, LO3	Completed
Literature Review	LO1, LO4, LO5	Completed
Requirement Elicitation	LO1, LO3, LO4, LO5, LO6	Completed
Design	LO3, LO4	Completed
Implementation	LO3, LO4, LO5, LO7	Completed

Testing and Evaluation	LO4, LO6, LO7, LO8	Completed

10.3. Utilization of Knowledge from the Course

Programming Principles I & II - With the knowledge from this course, the development programming language and tools for the project were selected. The model training and integrating models were also done using the knowledge gained from this module.

Software Development Group Project - The new technology addition to the modules of the course learned during the development of this project was helpful for documenting and developing this research.

Algorithms - To get the understanding of the algorithm used in the T5 model and train and finetune the model which is the core element of this research development.

Client Server Architecture and Computer Systems Fundamental - The knowledge gained from this module helped in integrating the components of the developed model with a server and frontend.

10.4. Use of Existing Skills

Machine learning - Understanding key concepts such as supervised and unsupervised learning, regression, classification, clustering helped to research on models and work with datasets.

Fullstack R&D – The experience gained during the internship helped in integrating the modules and components and completing the research implementation.

10.5. Use of New Skills

NLP – Gained knowledge in this field since the core component of the research is of this field and the model used was also a transformer based language model which is relevant to this research area.

Deep Learning – This was a completely new area to gain knowledge. The used model for training is a neural network model which requires proper knowledge to handle and train for code generation.

10.6. Achievement of Learning Outcomes

Table 37:Achievemnets of learning outcome

Research Objectives	**Learning Outcomes**	**Status**
Problem Identification	LO2, LO3	Completed
Literature Review	LO1, LO4, LO5	Completed
Requirement Elicitation	LO1, LO3, LO4, LO5, LO6	Completed
Design	LO3, LO4	Completed
Implementation	LO3, LO4, LO5, LO7	Completed
Testing and Evaluation	LO4, LO6, LO7, LO8	Completed

10.7. Problems and Challenges Faced

Table 38: Problems and challenges faced

Challenges	**Description**	**Approach**
Data Limitation	The tool's ability to accurately transcribe natural language commands into Quorum source code will depend on the amount and quality of training data used to train the model. If there is limited training data available, the tool may struggle to accurately transcribe more complex commands or commands with unique syntax.	Custom dataset with narrowed scope was defined to complete the project within the time frame.
GPU Limitation per day	Google Cloud Platform offers different GPU types (such as NVIDIA Tesla and T4) and service plans with different hourly usage limits, ranging from free usage up to hundreds or thousands of dollars per month for high-performance computing tasks.	Training period was prolonged due to the restricted GPU runtime and the normal runtime consumed more time to train. Dataset was trained into batches to overcome the challenge.

No similar models to generate Quorum source code to benchmark	For benchmarking there aren't any current models to generate Quorum source code for natural language commands.	Holdout validation was used to evaluate the models.
Hardware Limitation	Memory limitation, processing power, storage limitations and network limitations while training the models.	Dataset was divided into batches and trained for prolonged time period with the limited resources.

10.8. Deviations

10.8.1. Scope related Deviations

The scope was to create a simple syntax for the identified problem and provide and programming environment for it. After the requirement gathering process there is a deviation in the scope according to the requirements gathered from the various stakeholders related to this research. The accuracy of the predicting code that should be generated and ease of editing the code were the priority requirements identified from the requirement gathering process. Instead of the simple syntax the current scope is to increase voice to source code generation accuracy and provide a sophisticated code editing tool.

10.8.2. Schedule related deviations

According to the schedule LR should have been completed by the time but there is a deviation in it. The scheduled time for the LR was affected since there is a deviation in the identified scope and new scope was added after the requirement gathering. While aggregating and organizing the gathered requirements the priority of some scope changed and new scope was added. The planned timeline had deviations connected to this. There were major deviations in conducting the interviews since finding relevant stakeholders and scheduling interviews took more time than planned and due to unavoidable reasons there were scenarios to reschedule the interview sessions. There was a delay in collecting requirements from this particular technique. The selection of tools was redone due to change of the scope. In prototype implementation since custom dataset was created to generate code snippets the selected programming language doesn't have any existing data more time was consumed than scheduled to train the model. The other parts of the schedule was almost completed without deviations according to the Gantt chart given below.

10.9. Limitations of the Research

Navigation supported by voice commands is not included in this implementation stage of this research. Classes and actions with advanced syntax structures are not generated due to limited custom dataset. Multiple variations are restricted for classes and actions due to the time constraint of the project. Multi language-specific input commands are also not supported in this stage of development. Due to the limited dataset the scope of the syntax was defined to specify the tasks and use cases for which the Quorum language code will be generated. Types and variables, operators, output and user input, type casting, commenting code, conditional statements with if, repeat, loops and basic structures of classes and actions of Quorum syntax are included to the scope to generate source code in the custom dataset.

10.10. Future Enhancements

The model needs to be trained in a large dataset with variations for the model to understand the syntax and generate code for new and complex natural language commands. The natural language commands are now only for English language input as voice commands but should widen it to multilingual commands for code generation to provide more accessibility to start learning programming syntax and concepts without language barriers. The defined scope needs to be extended to cover all sections of syntax of Quorum programming language. The editor needs complete assistance by voice for accessibility to handle the tool. Voice assistance should be enabled for the code edition to make changes to the generated code.

10.11. Achievement of the contribution to body of knowledge

1.12.1. Contribution to the Research Domain

The contribution to the research field is a custom dataset to train the T5-small variant of the transformer architecture, a type of neural network created for natural language processing tasks, and the trained model to produce Quorum programming language source code for natural language commands.

1.12.2. Contribution to the Problem Domain

By enabling users to code using their speech, a voice to source code generation tool can significantly improve accessibility and inclusion. With a voice-based interface, those with physical limitations or impairments that make typing difficult or impossible can more easily access coding. Students with upper limb disabilities may find it inspiring to be able to write code using their voice because it will give them more confidence and assurance as they pursue their education in software development. By providing a voice-based coding tool, this technology can eliminate this barrier, enabling students with upper limb disabilities to participate in coding activities and even inspiring them to pursue careers in software development more successfully and faster.

10.12. Concluding Remarks

In conclusion, the project described in this document is aimed at generating Quorum source code from natural language instructions, specifically in the English language. The project scope includes the generation of basic Quorum code structures for types, variables, and control variables, while allowing for variations in the natural language instructions provided by the user. However, advanced syntax structures for classes and actions are out of scope, as well as support for navigation by voice commands and multi-language input.

Overall, the project represents a promising development in the field of natural language processing and programming languages. It has the potential to make programming more accessible and intuitive for a wider range of users, especially those who are not familiar with traditional programming languages. While there are limitations to the scope of the project, it still represents a significant achievement that could pave the way for further research and innovation in this area.

REFERENCES

A. Ahmad, K. L. C. F. S. M. A. A. Y. a. S. G., 2018. An Empirical Study of Investigating Mobile Applications Development Challenges. *IEEE Access.*

A. B. Nassif, I. S. I. A. M. A. a. K. S., 2019. *Speech Recognition Using Deep Neural Networks: A Systematic Review.* s.l., IEEE, pp. 19143 - 19165.

Adnan Rahim Khan, C. C. S. D. C. A. D. K. P. A. R. B., 2022. VOCAL PROGRAMMING: A HANDS-FREE APPROACH FOR PHYSICALLY CHALLENGED INDIVIDUALS. *International Research Journal of Modernization in Engineering Technology and Science.*

al, A. M. e., 2021. *Studying the Usage of Text-To-Text Transfer Transformer to Support Code-Related Tasks.* Madrid, ES, IEEE, pp. 336-347.

Apostolos, T., 2017. Oral Programming Interface Using Speech Recognition Technologies.

Ayushi Trivedi, N. P. P. S. S. a. S. A., 2018. Speech to text and text to speech recognition systems-Areview. *IOSR Journal of Computer Engineering (IOSR-JCE)* , pp. 36-43.

Bharathi, B. S. K. a. N. M., 2016. Real Time Speech Based Integrated Development Environment for C Program. *Scientific Research Publishing.*

Cody L. McDonald, C. L. B. D. K. R. &. K. M., 2019. Perceptions of ability among adults with upper limb absence: impacts of learning, identity, and community. *Disability and Rehabilitation.*

Colin Raffel, N. S. A. R. K. L. S. N. M. M. Y. Z. W. L. a. P. J. L., 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research,* pp. 1-67.

Davidson, J., 2004. A comparison of upper limb amputees and patients with upper limb injuries using the Disability of the Arm, Shoulder and Hand. *Disability and Rehabilitation,* Volume 26, pp. 917-923.

Deek, F., 2003. A Case Study in an Integrated Development and Problem Solving Environment. *Journal of Interactive Learning Research,* Volume Association for the Advancement of Computing in Education (AACE), pp. 333-359.

E. Dehaerne, J. V. a. C. R., 2020. Code Generation Using Machine Learning: A Systematic Review.. *Journal of Artificial Intelligence Research,* Volume 68, pp. 1-25.

F. H. Boot, J. O. D. M., 2018. Access to assistive technology for people with intellectual disabilities: a systematic review to identify barriers and facilitators. *Wiley Online Library.*

Francis, T. &., 2004. Programming Environments for Novices. In: *Computer Science Education Research.* s.l.:Taylor & Francis Group, p. 28.

Graham, A. B. a. S. L., 2006. An Assessment of a Speech-Based Programming Environment. *Visual Languages and Human-Centric Computing (VL/HCC'06),* pp. 116-120.

Holper, L., 2010. Characterization of functioning in multiple sclerosis using the ICF. *Journal of Neurology.*

Ilse Lamers, D. C. C. C. C. R. B., 2015. Associations of Upper Limb Disability. *Neurorehabilitation and Neural Repair,* Volume 29, pp. 1-10.

Joe Zhang, S. B. W. W. P. C. H. S. H. S. H. A. J. H. &. J. T. T., 2022. Moving towards vertically integrated artificial intelligence development. *npj Digital Medicine* .

Julio Abascal, A. A. I. C., 2011. *Automatically Generating Tailored Accessible User Interfaces for Ubiquitous Services.* Donostia, Spain, ASSETS.

Jung, H. et al., 2019. *TurtleTalk: An Educational Programming Game for Children with Voice User Interface.* Scotland Uk, s.n., pp. 1-6.

K. Rustan M. Leino, V. W., 2014. The Dafny Integrated Development Environment. *EPTCS,* pp. 3-15.

Kıvanç Muşlu, Y. B. R. H. M. D. E. D. N., 2012. Speculative analysis of integrated development environment recommendations. *ACM SIGPLAN Notices,* 47(10), p. 669–682.

Koulouri, T. L. S. &. M. R. D., 2014. Teaching Introductory Programming. *ACM Transactions on Computing Education,* pp. 1-28.

Lagergren, M. S. M., 2021. *Programming by voice: Efficiency in the Reactive and Imperative Paradigm,* s.l.: s.n.

Lili Mou, R. M. G. L. L. Z. Z. J., 2015. *On End-to-End Program Generation from User Intention.* Beijing, Association for Computing Machinery (ACM).

Ludi, O. O. &. S., 2022. *Helping Students with Motor Impairments Program via Voice-Enabled Block-Based Programming.* s.l., Springer, Cham, pp. 62-77.

M. G. Prakash, A. P. S. D. A. A. S. B. M. R. a. P. L., 2022. *VSCODE- Code With Voice Using Natural Language Processing(NLP).* Chennai, India, 2022 International Conference on Computer, Power and Communications (ICCPC).

M. T. Tausif, S. C. M. S. H. M. H. a. H. H., 2018. *Deep Learning Based Bangla Speech-to-Text Conversion.* Yonago, Japan, IEEE, pp. 49-54.

Miyazaki, R., 2019. Establishing document creation support for people with upper limb disabilities by hands-free speech recognition and gaze detection. pp. 67-69.

Murad, C. & Munteanu, C., 2020. *Jung, Hyunhoon; Kim, Hee Jae; So, Seongeun; Kim, Jinjoong; Oh, Changhoon (2019). [ACM Press Extended Abstracts of the 2019 CHI Conference - Glasgow, Scotland Uk (2019.05.04-2019.05.04)] Extended Abstracts of the 2019 CHI Conference on Human Fact.* s.l., s.n.

Nizzad, A. M., 2022. *Designing of a Voice-Based Programming IDE for Source Code Generation: A Machine Learning Approach.* s.l., IEEE.

Okafor, O., 2022. Helping students with cerebral palsy program via voice-enabled block-based programming. *ACM SIGACCESS Accessibility and Computing.*

Oza, K. S., Patil, S. R. & Kamat, R. K., 2015. *1 C Programming: Logical Continuum of Program of Programming, in 'C' Programming in Open Source Paradigm: A Hands on Approach.* s.l.: River Publishers.

Prakash M Nadkarni, 1. L. O.-M. W. W. C., 2011. Natural language processing: an introduction. *Journal of the American Medical Informatics Association (JAMIA),* Volume 18, pp. 544-551.

S. Hossain, M. A. E. M. H. M. R. Z. a. O., 2021. *Code Generator based on Voice Command for Multiple Programming Language.* Kharagpur, India, IEEE, pp. 01-05.

Shelke, Y. H. G. a. S. D., 2016. *Speech to text conversion for multilingual languages.* Melmaruvathur, India, IEEE, pp. 0236-0240.

Smutný, P., 2012. *Mobile development tools and cross-platform solutions.* Tatras, Slovakia, IEEE.

Tudor B.Ionescu, S. S., 2020. Programming cobots by voice: A human-centered, web-based approach.

Wagner, A., 2014. Empowering Motorically Challenged Students to Program by Voice.

Yuding Liang, K. Z., 2018. Automatic Generation of Text Descriptive Comments for Code Blocks. *Thirty-Second AAAI Conference on Artificial Intelligence* , p. 32.

Yue Kang, Z. C. C.-W. T. Q. H. &. H. L., 2020. Natural language processing (NLP) in management. *Journal of Management Analytics.*

Zhevaho, O. O., 2021. An Overview of Tools for Collecting Data on Software Development and Debugging Processes from Integrated Development Environments. *INFORMATION AND COMMUNICATION TECHNOLOGIES AND MATHEMATICAL MODELING.*

Appendix A – Gantt Chart

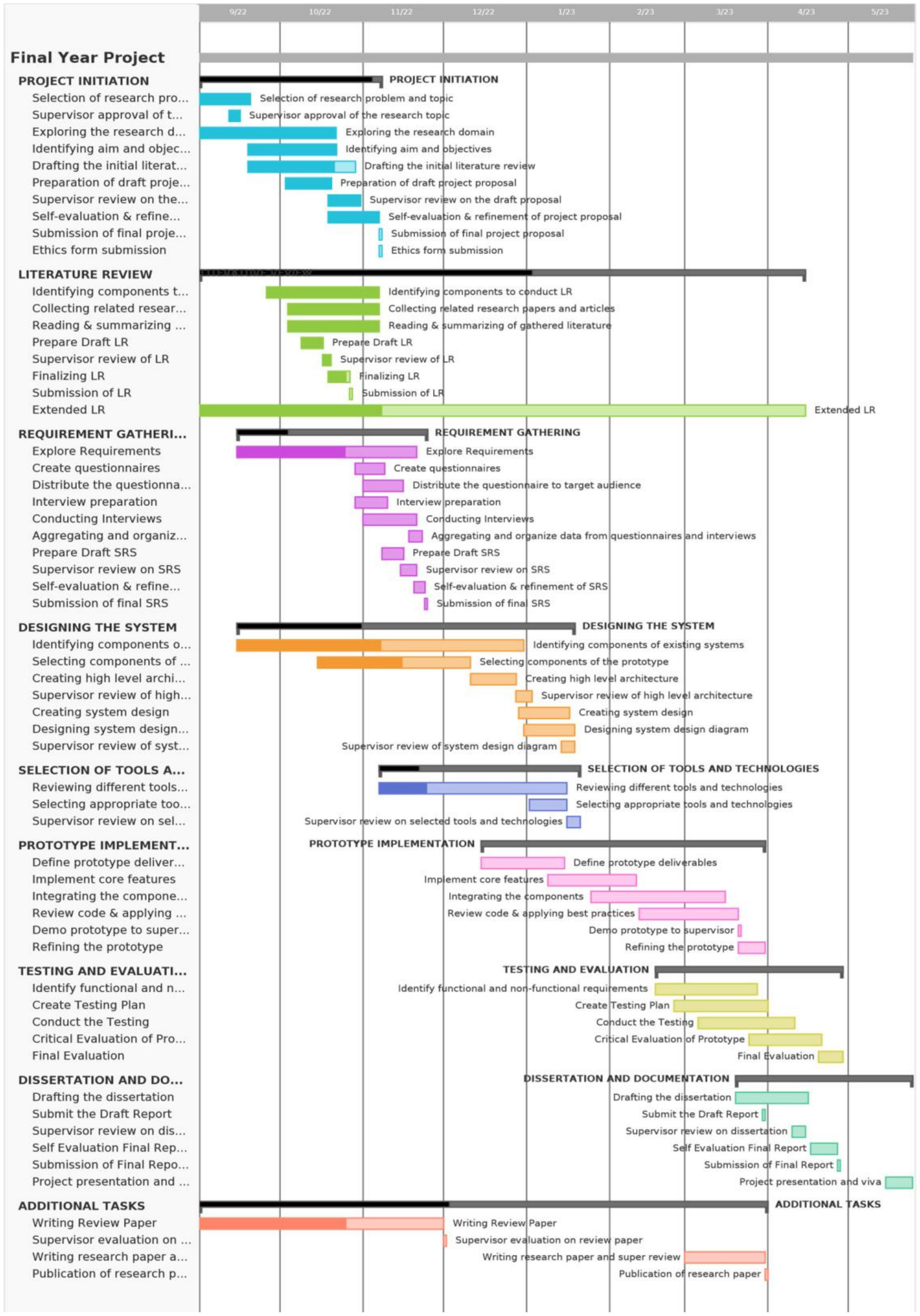


Figure 32:Gantt chart

Appendix B - SRS

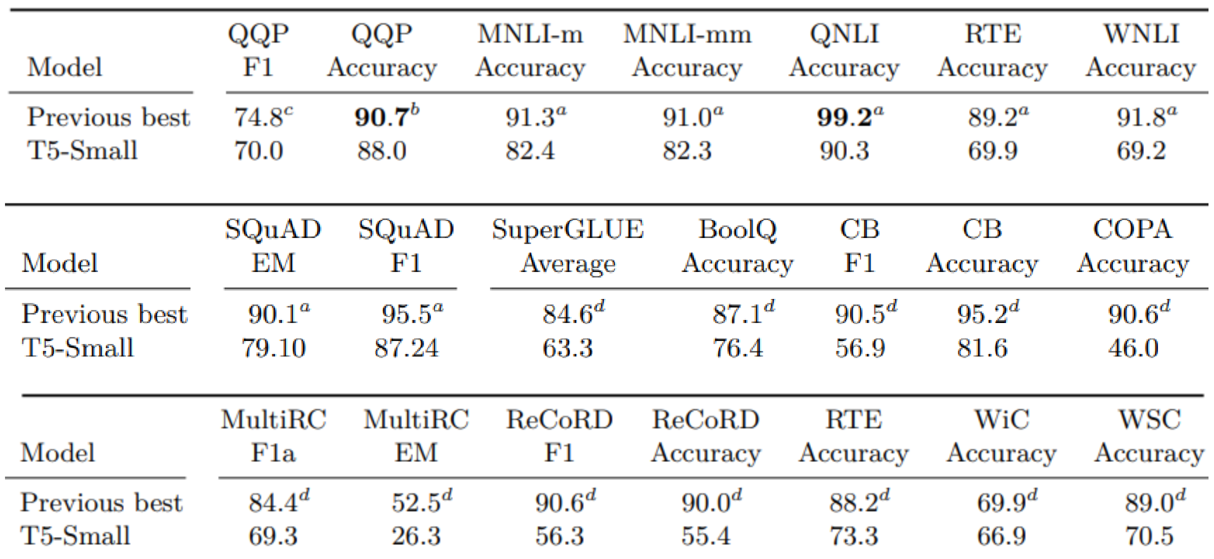
Table 39:Requirment gathering from stakeholders

Question	What are the main challenges faced by upper limb disabled students in accessing programming tools and software?
Targeted Stakeholder	UL Disabled Students
Findings	
Upper limb disabled students face several challenges, such as difficulties in typing or using a mouse, which can make it hard for them to access programming tools and software. They may also have difficulty with fine motor skills, which can make it hard to perform precise movements required in programming.	
Question	What are the main features that a voice to source code generation tool should have to meet the needs of upper limb disabled students?
Targeted Stakeholder	Experts
Findings	
A voice to source code generation tool for upper limb disabled students should have features such as voice recognition, natural language processing, and the ability to generate code in a format that is accessible and easy to use for students with disabilities. It should also have the ability to integrate with other assistive technologies, such as screen readers or speech synthesizers.	
Question	How can the tool be designed to ensure that it is accessible and easy to use for upper limb disabled students?
Targeted Stakeholder	UL Disabled Students
Findings	
The tool should be designed with accessibility in mind, such as ensuring that it is compatible with assistive technologies, providing alternative input methods such as voice recognition, and using clear and simple language in the user interface. It should also be tested with users with disabilities to ensure that it is usable and effective for them.	
Question	What are the potential benefits of using a voice to source code generation tool for upper limb disabled students?
Targeted Stakeholder	Experts

Findings	
The benefits of using such a tool include increased accessibility to programming tools and software for students with disabilities, improved productivity, and enhanced independence in programming. It can also provide a more inclusive learning environment and help bridge the digital divide for students with disabilities.	
Question	What are some potential challenges in implementing a voice to source code generation tool for upper limb disabled students?
Targeted Stakeholder	Experts
Findings	
Some potential challenges may include the need for specialized hardware or software, compatibility issues with existing systems, and the need for ongoing technical support and training. There may also be challenges related to privacy and security, particularly in the storage and processing of voice data.	

Appendix C – Design

Appendix C1 – Algorithmic Analysis

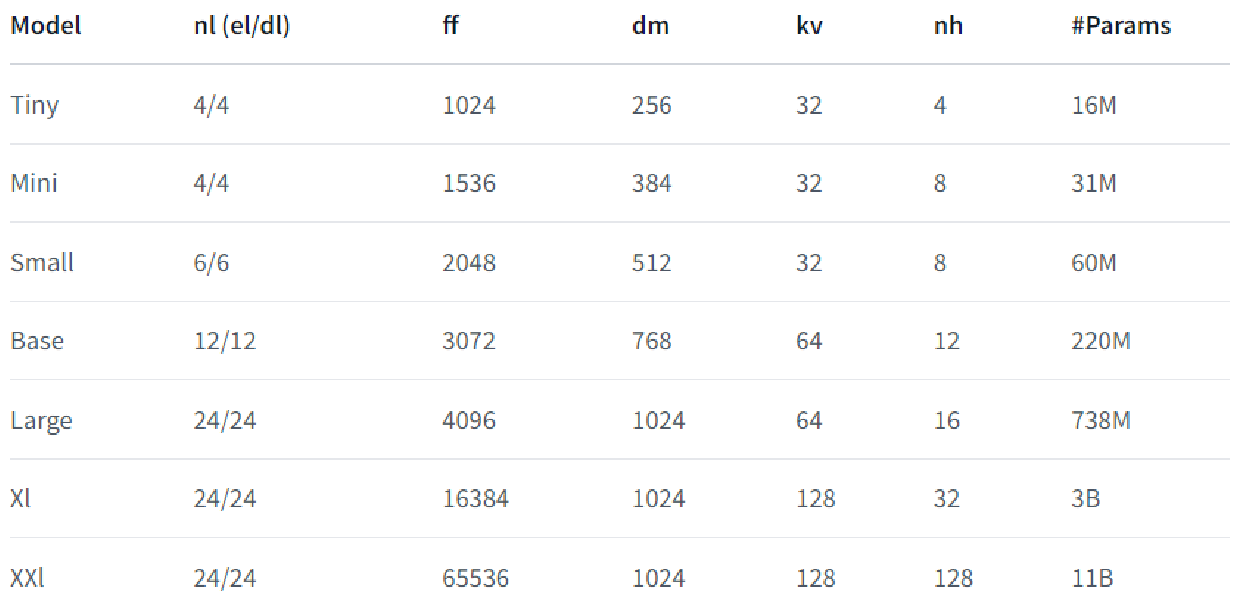


Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	74.8[c]	**90.7**[b]	91.3[a]	91.0[a]	**99.2**[a]	89.2[a]	91.8[a]
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2

Model	SQuAD EM	SQuAD F1	SuperGLUE Average	BoolQ Accuracy	CB F1	CB Accuracy	COPA Accuracy
Previous best	90.1[a]	95.5[a]	84.6[d]	87.1[d]	90.5[d]	95.2[d]	90.6[d]
T5-Small	79.10	87.24	63.3	76.4	56.9	81.6	46.0

Model	MultiRC F1a	MultiRC EM	ReCoRD F1	ReCoRD Accuracy	RTE Accuracy	WiC Accuracy	WSC Accuracy
Previous best	84.4[d]	52.5[d]	90.6[d]	90.0[d]	88.2[d]	69.9[d]	89.0[d]
T5-Small	69.3	26.3	56.3	55.4	73.3	66.9	70.5

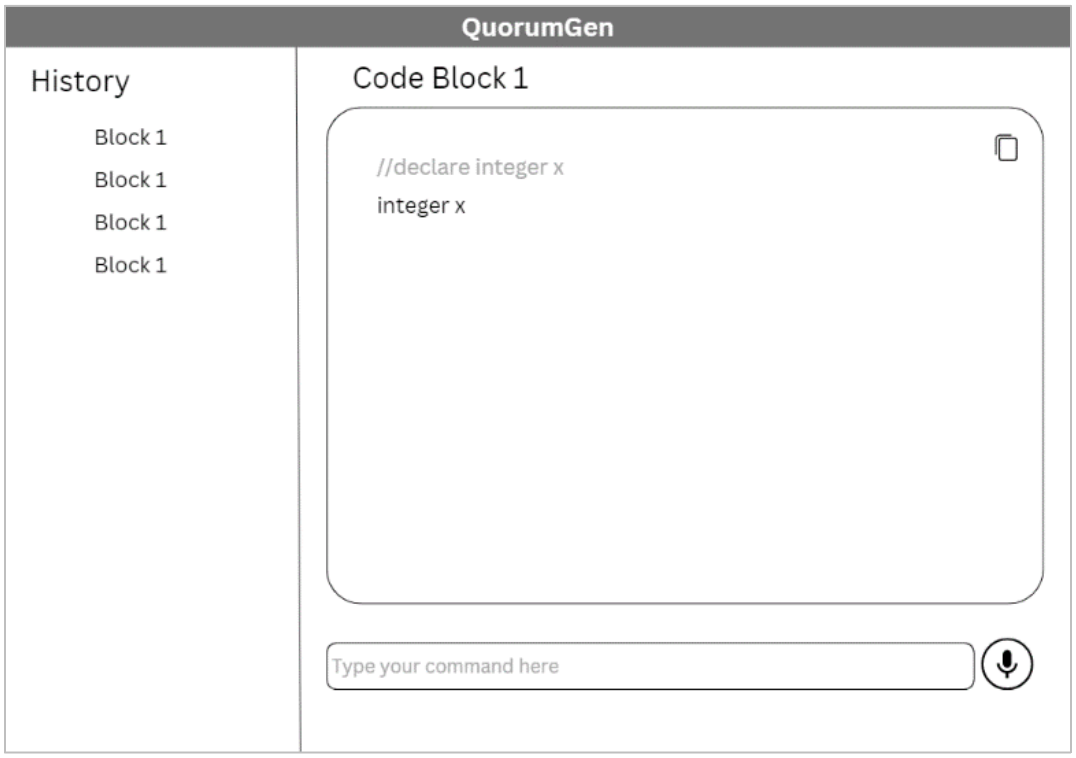
Figure 33: Algorithm analysis appendix



Model	nl (el/dl)	ff	dm	kv	nh	#Params
Tiny	4/4	1024	256	32	4	16M
Mini	4/4	1536	384	32	8	31M
Small	6/6	2048	512	32	8	60M
Base	12/12	3072	768	64	12	220M
Large	24/24	4096	1024	64	16	738M
Xl	24/24	16384	1024	128	32	3B
XXl	24/24	65536	1024	128	128	11B

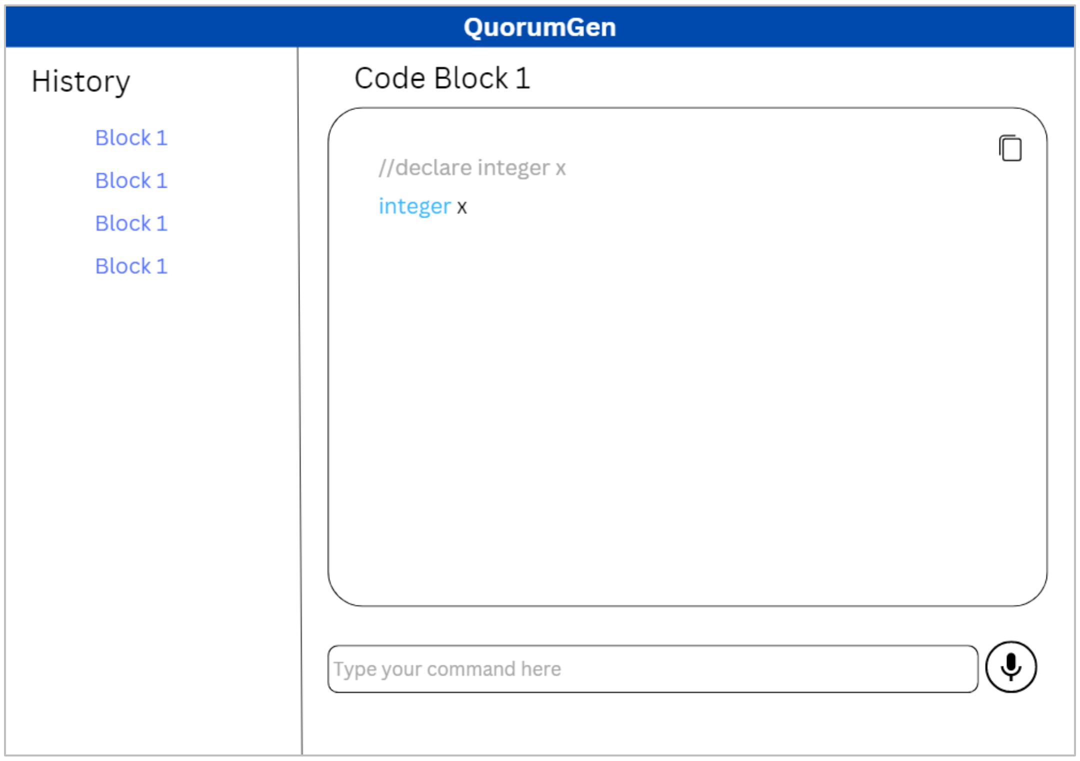
Figure 34: T5 variants hyperparameters

Appendix C2 – Algorithmic Analysis



```
//declare integer x
integer x
```

Figure 35:Low level fidelity wireframe diagram

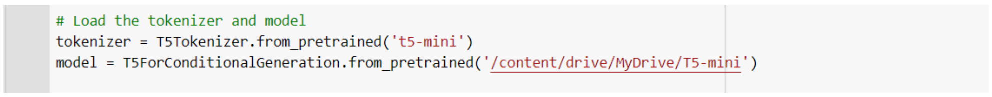


```
//declare integer x
integer x
```

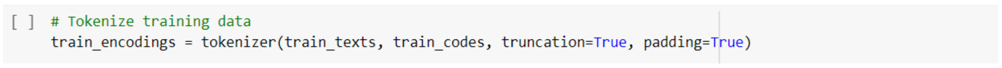
Figure 36:High level fidelity wireframe diagram

Appendix D – Testing

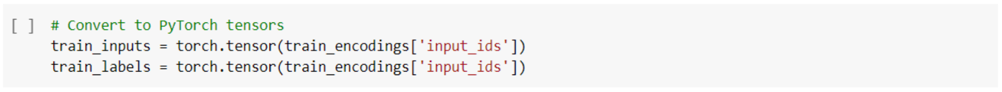
Appendix D1 - T5-mini Evaluation



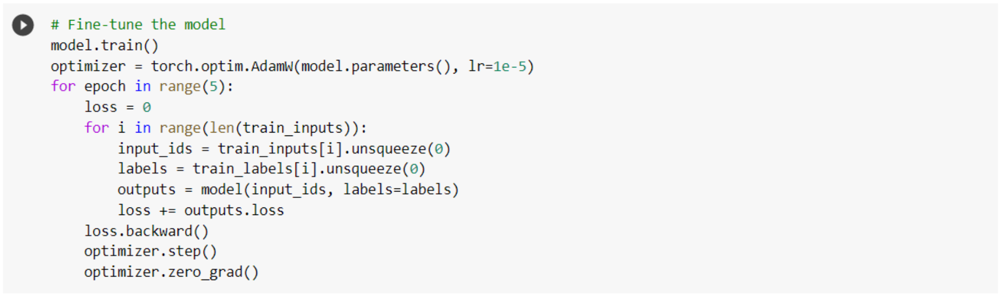
```
# Load the tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('t5-mini')
model = T5ForConditionalGeneration.from_pretrained('/content/drive/MyDrive/T5-mini')
```



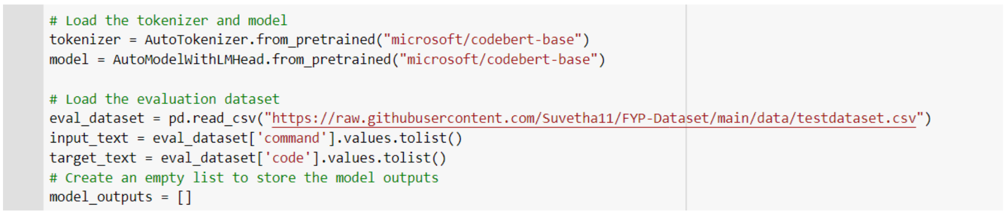
```
[ ]  # Tokenize training data
     train_encodings = tokenizer(train_texts, train_codes, truncation=True, padding=True)
```



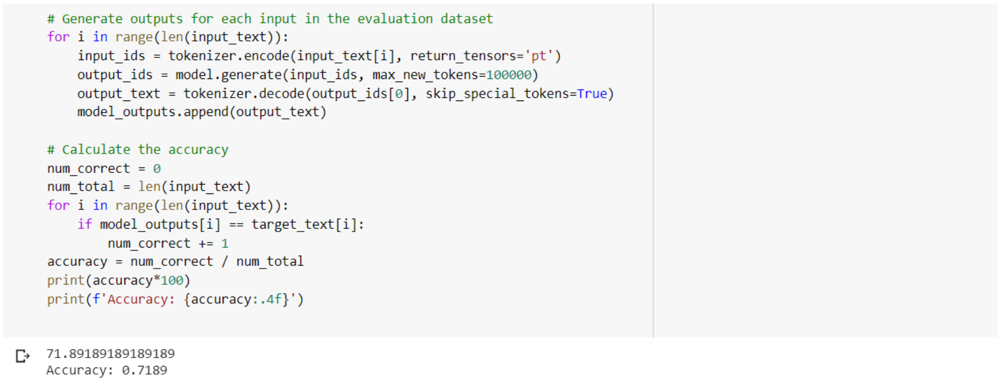
```
[ ]  # Convert to PyTorch tensors
     train_inputs = torch.tensor(train_encodings['input_ids'])
     train_labels = torch.tensor(train_encodings['input_ids'])
```



```
# Fine-tune the model
model.train()
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
for epoch in range(5):
    loss = 0
    for i in range(len(train_inputs)):
        input_ids = train_inputs[i].unsqueeze(0)
        labels = train_labels[i].unsqueeze(0)
        outputs = model(input_ids, labels=labels)
        loss += outputs.loss
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

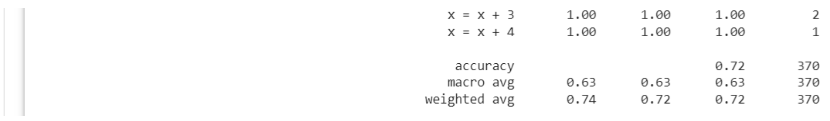


```
# Load the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("microsoft/codebert-base")
model = AutoModelWithLMHead.from_pretrained("microsoft/codebert-base")

# Load the evaluation dataset
eval_dataset = pd.read_csv("https://raw.githubusercontent.com/Suvetha11/FYP-Dataset/main/data/testdataset.csv")
input_text = eval_dataset['command'].values.tolist()
target_text = eval_dataset['code'].values.tolist()
# Create an empty list to store the model outputs
model_outputs = []
```



```
# Generate outputs for each input in the evaluation dataset
for i in range(len(input_text)):
    input_ids = tokenizer.encode(input_text[i], return_tensors='pt')
    output_ids = model.generate(input_ids, max_new_tokens=100000)
    output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    model_outputs.append(output_text)

# Calculate the accuracy
num_correct = 0
num_total = len(input_text)
for i in range(len(input_text)):
    if model_outputs[i] == target_text[i]:
        num_correct += 1
accuracy = num_correct / num_total
print(accuracy*100)
print(f'Accuracy: {accuracy:.4f}')
```

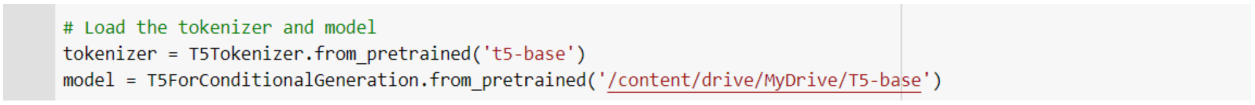
```
71.89189189189189
Accuracy: 0.7189
```



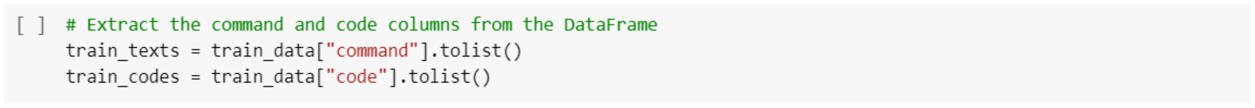
```
   x = x + 3       1.00      1.00      1.00         2
   x = x + 4       1.00      1.00      1.00         1

    accuracy                           0.72       370
   macro avg       0.63      0.63      0.63       370
weighted avg       0.74      0.72      0.72       370
```

Figure 37:T5-mini evaluation

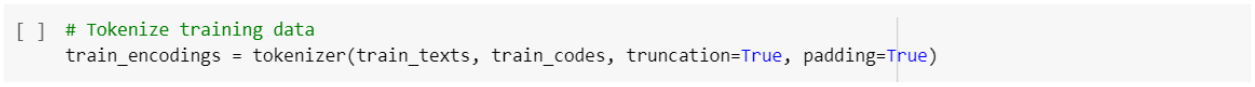
Appendix D2 - T5-mini Evaluation



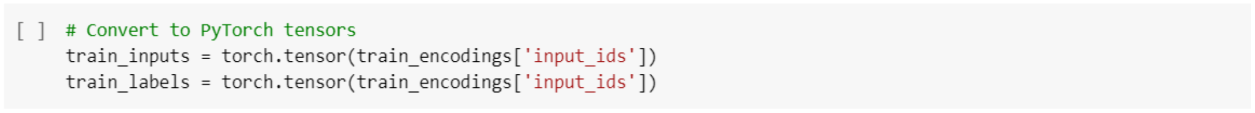
```
# Load the tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('t5-base')
model = T5ForConditionalGeneration.from_pretrained('/content/drive/MyDrive/T5-base')
```



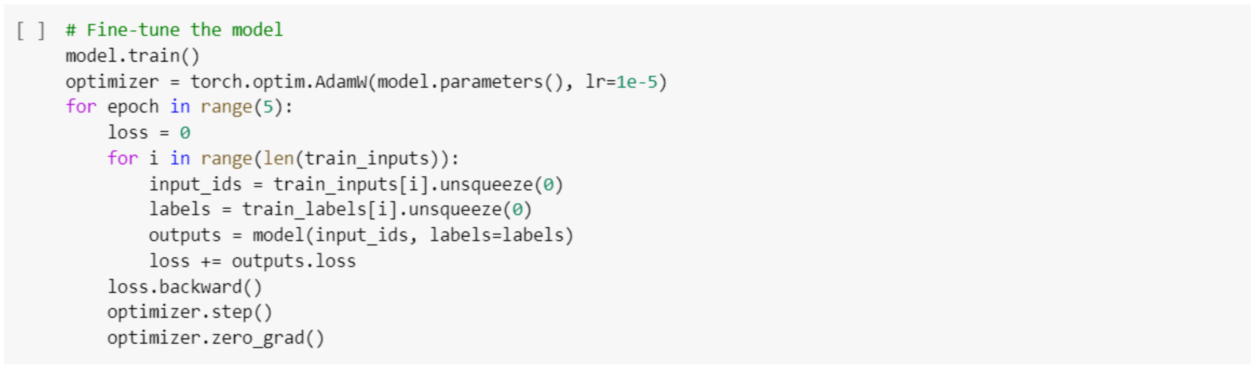
```
[ ] # Extract the command and code columns from the DataFrame
    train_texts = train_data["command"].tolist()
    train_codes = train_data["code"].tolist()
```



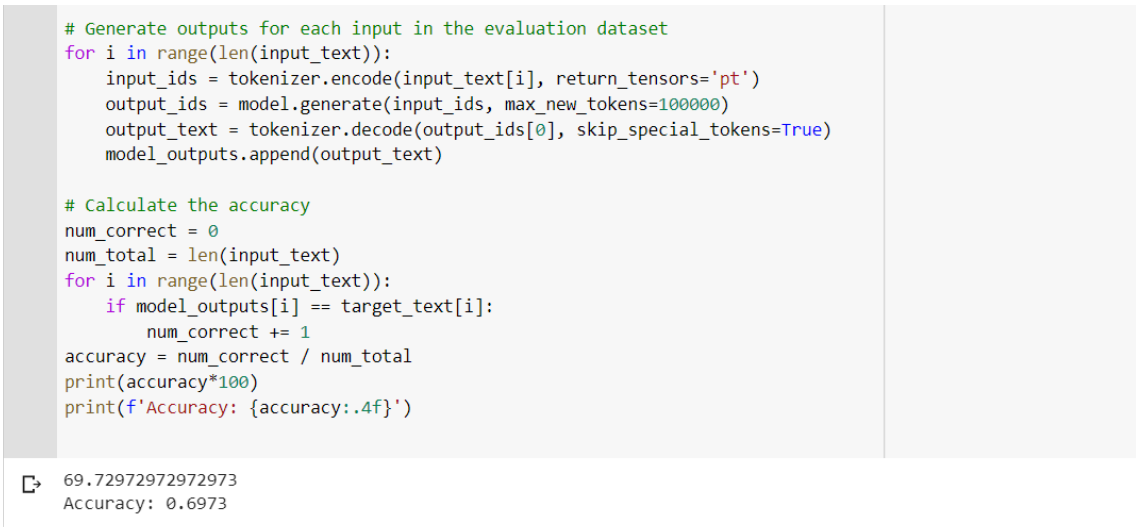
```
[ ] # Tokenize training data
    train_encodings = tokenizer(train_texts, train_codes, truncation=True, padding=True)
```



```
[ ] # Convert to PyTorch tensors
    train_inputs = torch.tensor(train_encodings['input_ids'])
    train_labels = torch.tensor(train_encodings['input_ids'])
```

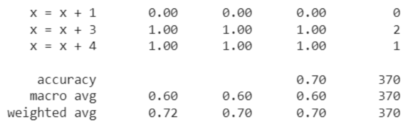


```
[ ] # Fine-tune the model
    model.train()
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
    for epoch in range(5):
        loss = 0
        for i in range(len(train_inputs)):
            input_ids = train_inputs[i].unsqueeze(0)
            labels = train_labels[i].unsqueeze(0)
            outputs = model(input_ids, labels=labels)
            loss += outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
```



```
# Generate outputs for each input in the evaluation dataset
for i in range(len(input_text)):
    input_ids = tokenizer.encode(input_text[i], return_tensors='pt')
    output_ids = model.generate(input_ids, max_new_tokens=100000)
    output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    model_outputs.append(output_text)

# Calculate the accuracy
num_correct = 0
num_total = len(input_text)
for i in range(len(input_text)):
    if model_outputs[i] == target_text[i]:
        num_correct += 1
accuracy = num_correct / num_total
print(accuracy*100)
print(f'Accuracy: {accuracy:.4f}')
```

```
69.72972972972973
Accuracy: 0.6973
```



```
   x = x + 1       0.00      0.00      0.00         0
   x = x + 3       1.00      1.00      1.00         2
   x = x + 4       1.00      1.00      1.00         1

    accuracy                           0.70       370
   macro avg       0.60      0.60      0.60       370
weighted avg       0.72      0.70      0.70       370
```

Figure 38:T5-base Evaluation

Appendix E - Evaluation

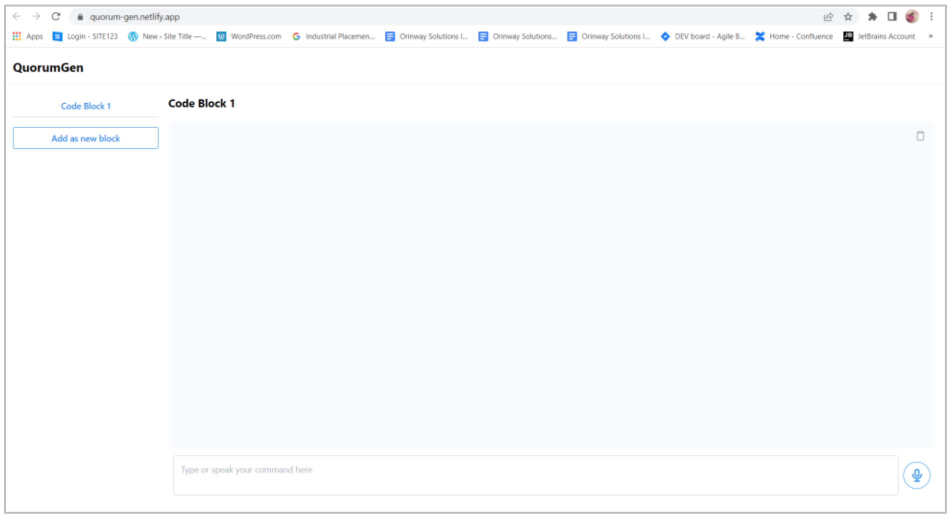
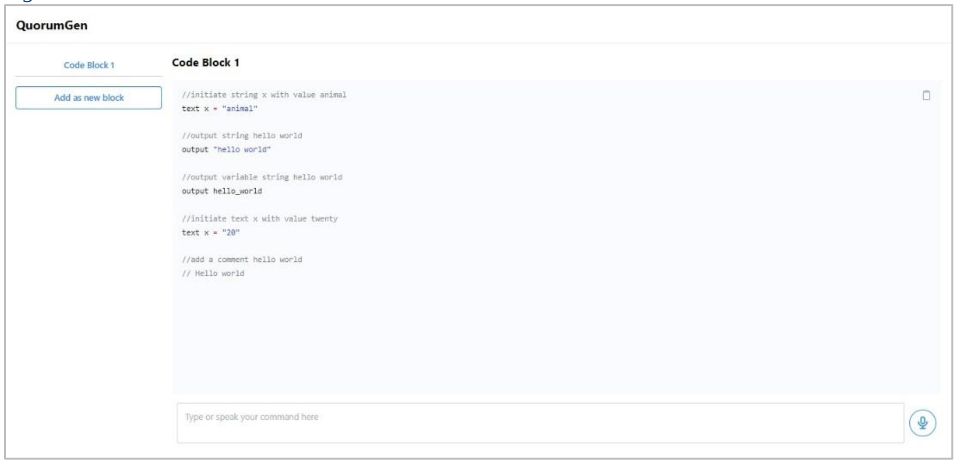
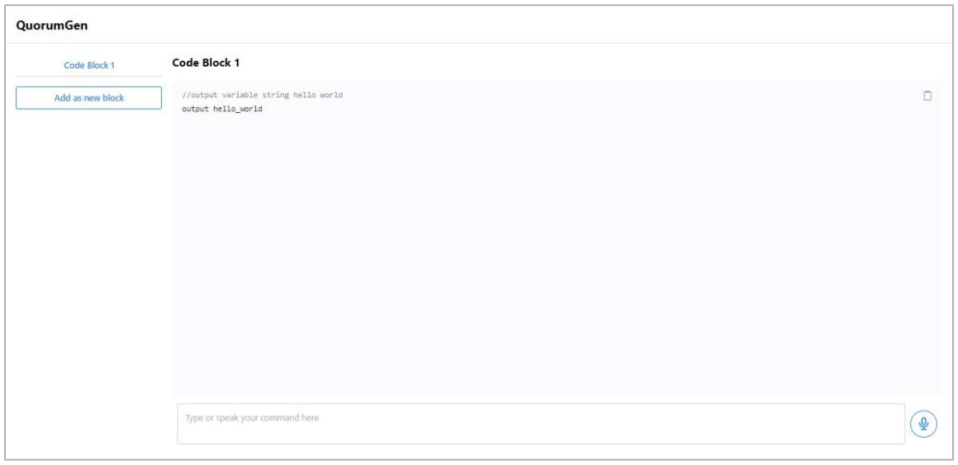


Figure 39:Tool Evaluation 1



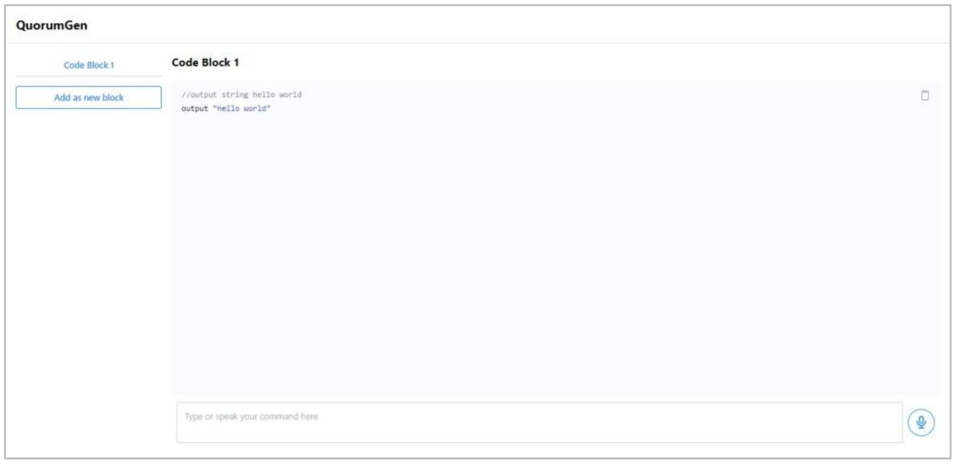
```
//initiate string x with value animal
text x = "animal"

//output string hello world
output "hello world"

//output variable string hello world
output hello_world

//initiate text x with value twenty
text x = "20"

//add a comment hello world
// Hello world
```

Figure 40:Tool Evaluation 2



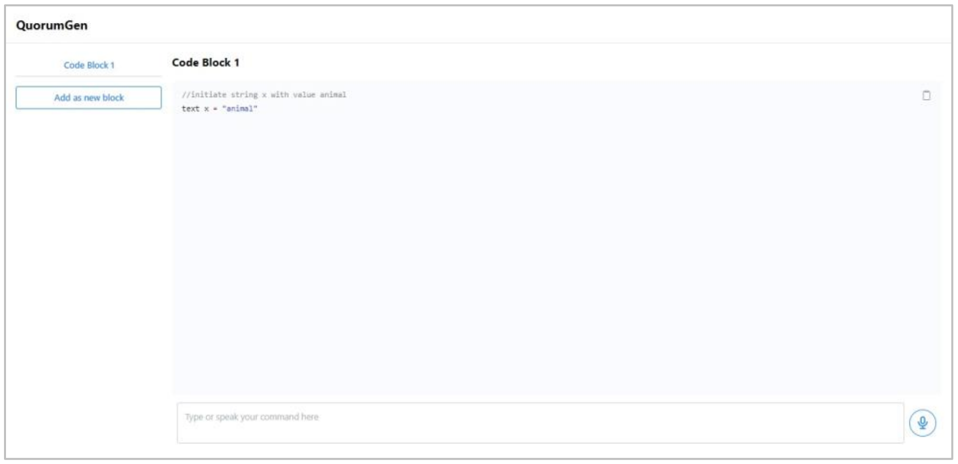
```
//output variable string hello world
output hello_world
```

Figure 41: Tool Evaluation 3



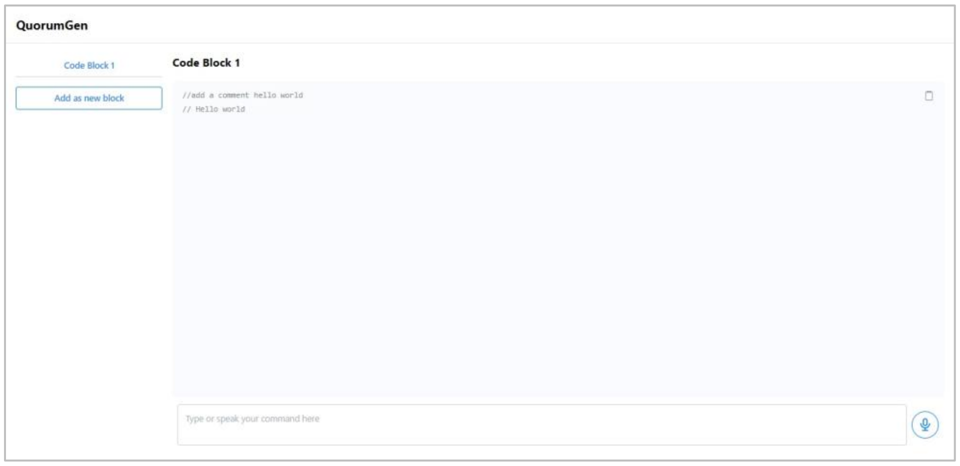
```
//output string hello world
output "hello world"
```

Figure 42:Tool Evaluation 4



```
//initiate string x with value animal
text x = "animal"
```

Figure 43:Tool Evaluation 5



```
//add a comment hello world
// Hello world
```

Figure 44:Tool Evaluation 6