INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

Detecting Code Quality Issues in AI-generated code: A deep learning approach

A Product Specification, Design and Prototype by

Mr Muhammad Areeb Niyas

W1809939 / 20200129

Supervised by

Mr Jihan Jeeth

Submitted in partial fulfilment of the requirements for the BSc(Hons) in Computer Science degree at the University of Westminster.

Date: February 2024

Abstract

The current tech era is rapidly evolving due to generative AI. The increase in AI- generated code in the industry rises questions about the nature and quality of AI-assisted programming. Especially when the future is headed towards; the majority of code in the world being AI-generated. Currently developed code analysis tools are built using object-oriented metrics and templates that check against a pre-set of coding rules and guidelines. Majority of the code quality tools developed still have limitations and constraints in their ability and they are built on top of code that is not AI-generated.

To adapt to the era of generative AI, a novel approach is to design and execute a CNN trained on AI-generated code. Recent research has demonstrated effectiveness in using deep learning techniques to detect code smells and software vulnerabilities accurately. This project would be the first in its nature to apply deep learning in identifying code quality issues particularly with AI-generated code.

The initial test results indicated that the proposed system successfully detected code quality issues by making predictions and it was particularly strong for detecting no issues and the ‘MultipleVariableDeclarations’ issue with good true positive and true negative values. However, for the other issues it was less reliable, however the initial implementation has the potential to be a groundbreaking novelty with the proposed improvements.

Subject Descriptors:

- Computing methodologies → Artificial intelligence
- Computing methodologies → Machine learning → Machine learning approaches → Neural networks
- Software and its engineering→ Software notations and tools→ Software maintenance tools
- Security and privacy→ Software and application security→ Software security engineering

Keywords: Code Quality, AI-generated code, AI, Deep Learning, Convolutional Neural Network

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
DL	Deep Learning
ML	Machine Learning
ANN	Artificial Neural Networks
RNN	Recurrent Neural Networks
CNN	Convolutional Neural Networks
LLM	Large Language Model

CHAPTER 01: INTRODUCTION

1.1 Chapter Overview

This chapter provides an overview of the project that has been undertaken by the author. First, the author will be diving into the project background in detail and discuss the problem. Then the existing work and its limitations will be discussed which leads to the research motivation along with the challenges faced, aims and objectives of this research project.

In this research project, the author tries to identify a novel deep learning approach for the code quality issues that can be highlighted in AI-Generated code and to introduce a novel system to detect these issues prior to a developer's implementation.

1.2 Problem Background

The current tech era is rapidly evolving, especially with generative AI. This is mainly because of the potential and ability of AI to carry out tasks that require high skill more reliably and quickly than humans (Harding et al., 2023).

Code generation is being widely used and spread with many generative tools which can be accessed and integrated seamlessly into the daily lives of a programmer. With the widespread and ease of use due to the outstanding performance, code generating models are released into the market swiftly in this era (Barke, James and Polikarpova, 2023). Therefore, the increase in AI-generated code in the industry gives rise to questions about the nature and quality of AI-assisted programming. Especially when the future is headed towards; the majority of code in the world being AI-generated.

1.2.1 AI-generated Code

ChatGPT and generative AI experienced a quick rise in daily use and popularity since its release in November 2022. Consequently, it has helped a lot of people with its amazing ability with human-like responses. The GPT 3.5 has shown significant potential for redefining a number of research domains including code generation.

The productivity of developers increases with automatic code generation by producing or completing the source/executable code automatically based on programming patterns or specifications.

However, issues concerning potential hazards are prevalent with the widespread and adoption of AI-Generated code. This is due to the lack of studies which formally investigate the reliability and quality of the code generated (Y Liu et al., 2023).

1.2.2 Code Quality Issues

AI-generated code especially ChatGPT commonly contains quality issues such as incorrect outputs, underperforming, maintainability and compilation errors (Y Liu et al., 2023). While many existing tools attempt to solve code quality issues and highlight common mistakes, current code quality tools in the industry do not use datasets from AI-generated code and code quality tools have not been built to target AI-generated code. Some common code quality issues include: syntax errors, duplicates, unsecure code, incorrect output and overly complex code.

1.2.3 Deep Learning

Machine Learning (ML) is a vital component in today's world of artificial intelligence which uses algorithms to imitate and find patterns in data, similar to how humans learn. Deep Learning (DL) is a subset of machine learning which uses multi layered neural networks which ideally tries to imitate the way a human brain functions. It has become very popular over the years due to the increase and widespread of big data and performing better than traditional ML techniques (Alzubaidi et al., 2021).

While machine learning methods have been used extensively to identify issues in code, recent research using deep learning methods, such as convolutional neural networks (CNNs), artificial neural networks (ANNs), and recurrent neural networks (RNNs), has shown a higher accuracy rate in detecting code smells when compared to traditional ML methods (Ho et al., 2023). However, code quality detectors targeting AI-generated code or built on top of less natural language text LLMs to avoid biases are not available. So, the author mainly considers using deep learning approaches to identify code quality issues in AI-generated code in this research as the risk of attacks and maintainability of code is a rising concern in the near future as developers enter an era of AI-generated code.

1.3 Problem Definition

There is a gap between an experience coder compared to generated code and current LLMs cannot entirely replace professional developers. In a prominent level, LLMs currently have a gap in identifying requirements and context of a problem clearly (Wu, 2023).

AI-generated code is frequently used in modern software development to automate repetitive operations and increase productivity. However, AI-generated code may experience a number of quality problems that result in bugs, inefficiencies, and difficulties (Liu et al., 2023).Therefore, consequently there is an increasing demand for automated tools that can evaluate the quality of code produced by AI models like ChatGPT.

A deep learning approach that provides a solution for the breakdown of above-mentioned problems is required to ensure a novelty prototype. This study therefore would aim to create a tool that can be used by developers seamlessly to identify code quality issues in AI-generated code prior to deployment.

1.3.1 Problem Statement

The majority of code written in the future will be AI-generated but there are code quality issues in such code and the current tools are not accurate analyzers therefore new novel systems are required to detect code quality issues catered to AI-specific code.

1.4 Aims and Objectives

1.4.1 Aims

The aim of this research is to design, develop and evaluate a novel analysis tool that will provide relevant, up-to-date, and accurate highlighting of code quality issues in AI-generated code by using machine learning and deep learning techniques.

To further elaborate, this research project will aim to provide a system that can be used by developers where they can input the AI-generated code and test for any code quality issues in the generated code. The use of data mining techniques, deep learning (DL) methods, data analysis, machine learning techniques will be researched to make the best possible analysis.

Further investigation to provide a prompt based on feedback from the analysis tool will also be explored. The required knowledge and expertise will be researched and the tool will be thoroughly tested to evaluate the accuracy.

1.4.2 Research Objectives

Research Objectives	Explanation	Learning Outcome	Research Questions
Problem Identification	Carry out thorough preliminaryresearch. **RO1**: To conduct thorough research on previous work to get a proper understanding ofthe problem.	LO1, LO2	RQ1, RQ3

	RO2: To evaluate current methodologies used to solve theproblem.		
Literature Review	Do an in-depth study on existing solutions and how deep learning and machine learning can be used to build a more accurate system. **RO3**: To learn about current existing analysis tools and solutions. **RO4**: To compare and evaluate current tools with AI-generated code. **RO5**: To discover limitations of existing analysis tools.	LO1, LO4, LO8	RQ1, RQ2,RQ3
Data Gathering and Analysis	**RO6:** To gather information about the need for analysis toolsfor AI-Generated code. **RO7:** To collect requirements for user expectations for an analysis system that would detect code quality issues. **RO8:** To get feedback and opinions from domain experts to build a good overall system.	LO2,LO3	RQ1, RQ2,RQ3
Research Design	**RO9:** To state and identify the design objectives.	LO1,LO5, LO8	RQ1, RQ3

	R10: To create a high-level architecture diagram to analyzethe structure. **R11:** To create a data flow diagram to evaluate the flow.		
Implementation	**R12:** To select the most suited technologies and tools for implementation. **R13:** To identify and create an accurate model that would detect multiple code quality issues in AI-generated code. **R14:** To make a user interface that would have good UX and integrate the core functionality	LO1, LO5, LO7, LO8	RQ1, RQ2,RQ3
Testing and Evaluation	**R15:** To create appropriate testplans for the user interface andthe model. **R16:** To utilize appropriate methods, evaluate the model and the user interface and document any limitations	LO5,LO8	RQ2
Publish Findings	Create well-organized documentation, reports, and articles that analyze the research findings critically. **R17**: To publish a research paper based on findings.	LO4,LO8	RQ1, RQ2,RQ3

	R18: To publish analysis and test results identified through the research. **R19**: To make the code or model open source so that furtherresearch may be pursued.		

Table 1: Research Objectives

1.5 Novelty of the Research

1.5.1 Problem Novelty

Even though code generations tools are starting to become widely used, the quality of the nature of AI-assisted programming have risen rapidly(Yetiştiren et al., 2023). Multiple code quality issues and code smells were reported from code generated through ChatGPT, Copilot and other generative AI tools. Current static analysis tools are built on top templates where the source code is checked against a pre-set of defined coding rules and guidelines and not built on top of AI-generated code (Pecorelli et al., 2019).

Existing tools and implementations face limitations and constraints. In some cases, it was found that even the defined templates has issues when it was tested against online programming tasks (Birillo et al., 2023). Recent studies have proven that deep learning approaches produce a higher accuracy in detecting code smells compared to traditional ML approaches and there is space for improvement (Ho et al., 2023).

However, detecting code smells has a comparatively less validity threat compared to detecting code quality issues (Mamun et al., 2019). Therefore, it is essential to detect code quality issues prior to deployment and the novelty of using deep learning to accurately detect code quality issues remains. Moreover, no studies as of yet focusses on the code quality issues generated by AI (Liu et al., 2023) taking it a step further in ensuring novelty.

1.5.2 Solution Novelty

There has been almost no work to identify code quality issues using deep learning methods (Sengamedu and Zhao, 2022). Deep learning has been explored in other areas such as software vulnerabilities and code smells (Das, Yadav and Dhal, 2019), however it is yet to be explored in the purpose of detecting code quality issues in AI-generated code.

1.6 Research Gap

Current code analysis tools analyze code against a pre-set of coding rules and validates according to given guidelines. This helps identify subtle defects or vulnerabilities. Low quality code can cause many problems in software development due to maintainability problems, security issues and inefficient code. Therefore, developers need to pay attention and take precautions about code quality prior to deployment (Vable, Diehl and Glymour, 2021). Recently, LLMs which contain much more natural language text than executable code (industry standard code) is used in code generation systems. This leads to biases which can reduce the quality of generated code (Mouselinos, Malinowski and Michalewski, 2023).

There has been almost no work to identify code quality issues using deep learning methods (Sengamedu and Zhao, 2022) which emphasizes the gap and the need to bridge it. While state-of-the-art code generation tools such as ChatGPT can generate effective code, the AI-generated code commonly contains quality issues such as incorrect outputs, underperforming, maintainability and compilation errors (Y Liu et al., 2023). So, code quality issues in AI-generated code is a critical issue in the field of code generation and no studies have been conducted to detect code quality issues specific to AI-generated code. Thus, a deep learning approach can be developed to fill this gap.

1.7 Contribution to the Body of Knowledge

The contributions to the body of knowledge will be achieved by creating a novel solution which is a deep learning-based model to identify code quality issues in AI-generated code before the code goes into deployment.

1.7.1 Contribution to Research Domain

While machine learning (ML) methods have been used extensively to identify issues in code, recent research using deep learning methods, such as convolutional neural networks (CNNs), artificial neural networks (ANNs), and recurrent neural networks (RNNs), has shown a higher accuracy rate in detecting code smells when compared to traditional ML methods (Ho et al., 2023). Therefore, it is proposed to explore a deep learning based approach in order to identify code quality issues similarly to how it was applied to detecting code smells accurately and additionally targeting AI-generated code.

1.7.2 Contribution to Problem Domain

Currently available solutions are primarily focused and built on top of code which is not AI-generated. Majority of the code quality tools developed still have limitations and constraints in their ability (Das, Yadav and Dhal, 2019). Therefore, a tool that highlights code quality issues targeting AI-generated code that hasn't been done before in previous research will be investigated so that developers can detect such issues prior to deployment.

1.8 Research Challenge

This research project's primary objective is to improve the accuracy and widespread use of deep learning techniques in code quality tools while overcoming the limitations in literature in this new code generation domain. The following is a list of research challenges:

1. **Finding research materials** – Due to the constant changes in the code generation domain it will be highly difficult to stay up to date on new case studies and constant changes. New techniques for data preparation should also be incorporated in order to raise the standard of publicly available datasets (Sharma, Sinha and Sharma, 2022). It is highly difficult to gather datasets that would provide successful results in the domain of using deep learning to detect code quality issues (Raychev, 2021).

2. **Developing and improving accuracy of deep learning model** – Using deep learning is a new approach in the domain of code quality. Even though machine learning has great potential in solving the issue of code quality, there has not been many successful outcomes as of yet (Raychev, 2021). The volume of data that specific ML algorithms can handle as well as specific data type limitations that some ML algorithms impose is a key challenge (Juddoo and George, 2020).

3. **Code quality analysis for AI-generated code** – At the point of initiation of this research project, there is no research that standardly investigates the quality of code and reliability of AI-generated code (Y Liu et al., 2023). Current static analysis tools in the industry are built using a variety of different object-oriented metrics and templates that check against a preset of coding rules and guidelines.

According to these challenges, it will be complex to construct a prototype that uses artificial intelligence to detect issues in code quality in AI-generated code.

1.8.1 Research Questions

RQ1: How can a DL approach be used to detect code quality issues in AI-generated code?

RQ2: How effective would a DL approach be in detecting code quality issues in AI-generated code compared to other tools?

RQ3: What are the latest advancements in deep learning that can be used to perform code analysis?

1.9 Chapter Summary

This chapter covered a comprehensive overview of the problem domain and background of the proposed solution which is to build a novel code analysis tool catered to address issues in AI-generated code. The research gap, aims and objectives being addressed along with its upcoming challenges were discussed in detail. The methodologies used will be discussed in the following chapter.

CHAPTER 02: SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Chapter Overview

This chapter focuses on the identification and steps of gathering requirements. Specifically, potential stakeholders along with their interaction points and roles are captured in a rich picture diagram and a stakeholder onion model. In addition, the requirement-gathering techniques and the insights acquired in order to analyze and generate functional and non-functional requirements, use case diagrams, and prototype descriptions are described.

2.2 Rich Picture Diagram

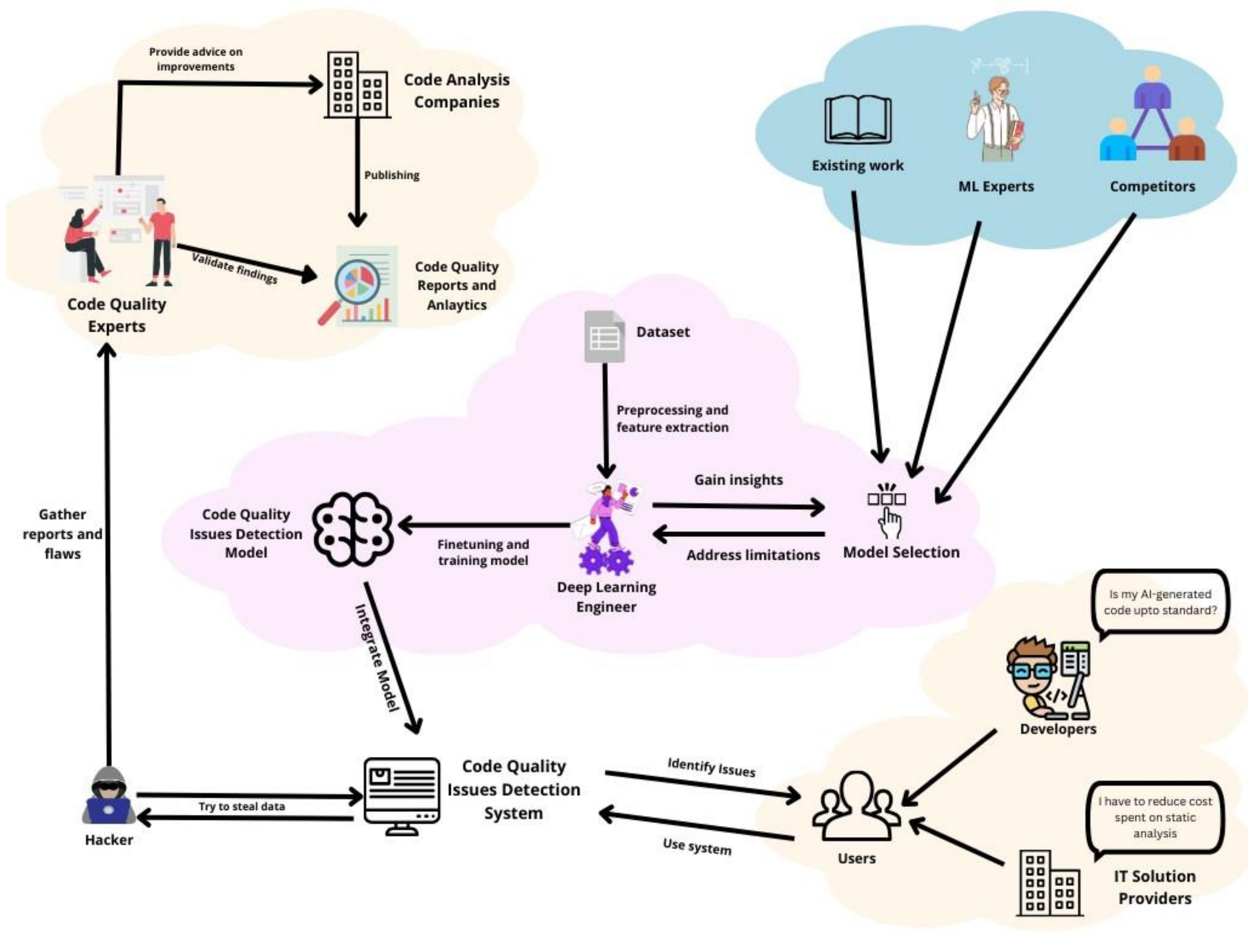


Figure 1: Rich Picture Diagram (self-composed)

The above diagram shows the identified structure, process and concerns of the proposed system which are visualized in the rich picture diagram. It is a way of seeing an overview of the project using a panoramic view and can be used to identify the stakeholders of the project.

2.3 Stakeholder Analysis

2.3.1 Stakeholder onion model

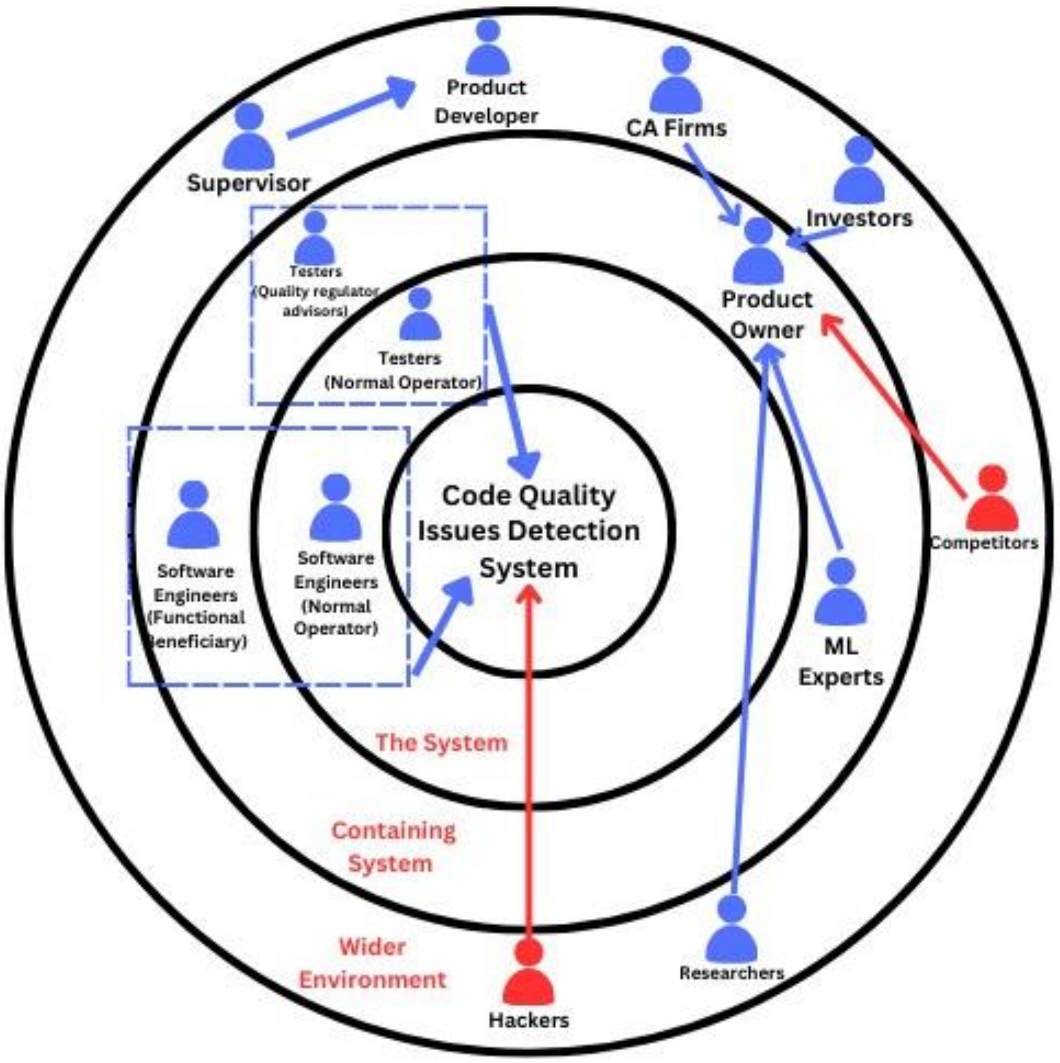

Product Developer
CA Firms
Supervisor
Investors
Testers (Quality regulator, advisors)
Testers (Normal Operator)
Product Owner
Code Quality Issues Detection System
Software Engineers (Functional Beneficiary)
Software Engineers (Normal Operator)
Competitors
ML Experts
The System
Containing System
Wider Environment
Researchers
Hackers

Figure 2: Stakeholder Onion Model (self-composed)

2.3.2 Stakeholder viewpoints

Stakeholders	Roles	Description
Software Engineers	Normal Operator	Perform code analysis using the system and identify code quality issues prior to deployment.
Testers	Quality regulator advisors/Normal operator	Evaluate the output and functionality of the system. On the other hand, be a normal operation when detecting code quality issues prior to deployment.
Product Owner	Operational beneficiary	Manage and oversee business processes and base product decisions on input from other stakeholders.
ML Experts	Functional beneficiary	Provide insights and feedback on the research domain and improve the system further by suggesting enhanced approaches of the core functionalities.
Code Analysis Firms		Provide insights and feedback on common code quality issues and give insights on current ways of implementation and gaps.
Product Developer	Maintenance	Develop and maintain the functionalities of the system.
Supervisor	Quality regulator/Advisor	Evaluate and provide constant feedback for useful suggestions and improvements.
Researchers	Functional beneficiary	Use the system to identify gaps and provide literature insights.
Competitors	Negative Stakeholder	Other parties that build similar products that may outperform the proposed system.
Hackers		Try to infiltrate and compromise the system

		making it vulnerable through unethical approaches.

Table 2: Stakeholder Analysis

2.4 Selection of Requirement Elicitation Methodologies

Requirements for a research project can be established with the requirement elicitation methodologies. The identified methodologies for this project: LR, interviews, and brainstorming are shown below along with the descriptive justifications applicable.

Method 1: Literature Review
Literature review is a technique for obtaining information about existing solutions which is used in research. It involves looking and studying research papers, articles and publications regarding AI-generated code and existing systems to evaluate and identify gaps. It is one of the crucial steps during code quality detection and creating well informed approaches that work. Therefore, the author carried out a comprehensive research and analyzed existing solutions and evaluated technologies to solve current limitations.
Method 2: Interviews
Carrying out interviews with experts such as technical leads and machine learning experts is a methodology which is crucial to identify the frequency of code quality issues and figure out the best practices for detection. This allows the author to gain insights from technical experts qualitatively which is crucial for this project more than quantitative data because the individuals in charge of code reviews and code analysis in organizations are usually leads and experts. Therefore, getting qualitative data from them to offload their workload would be one of the main objectives of this project.
Method 3: Brainstorming
Alternatively, brainstorming with other researchers and software developers involves a set of researchers and developers created to participate in the process of information sharing and detection of code quality issues in a software system and the development of mitigative measures. The aim is to produce a number of ideas and solutions along with identifying any possible issues in code quality detection that could have been missed during the early stages.

Table 3: Requirement Elicitation Methods

2.5 Discussion of Findings

2.5.1 Literature Review

Citation	Findings
(Yetiştiren et al., 2023)	Even though code generation tools are starting to become widely used, their output remains unreliable and undetermined. It was noticed that there were code quality issues in code generated through ChatGPT, Amazon CodeWhisperer and GitHub Copilot in different levels of severity which resulted in technical debt.
(Sengamedu and Zhao, 2022)	There has been almost no work to identify code quality issues in AI-generated code using deep learning. Neural language models can be built to detect code quality issues with less false positives.
(Ho et al., 2023)	LSTMs and BiLSTMs can be added to neural networks to improve the accuracy of neural networks when detecting code quality issues.
(Das, Yadav and Dhal, 2019)	Deep learning can be used to identify code smells. CNN models are accurate in identifying brain class and brain method code smells.
(Kanan, Sherief and Abdelmoez, 2022)	Deep learning approaches produce a higher accuracy in detecting code smells when compared to traditional ML approaches.

Table 4: Literature Review Findings

2.5.2 Interviews

The findings of the interview have been shown below where a thematic analysis was conducted. The list of interviewees and questions for the interview along with its purpose and mapped research questions are shown in Appendix A.

Codes	**Theme**	**Analysis**
Analysis tools	Current trends	The majority of participants use only in-built static analysis tools of the IDE for detecting code quality issues. Some use Sonar and Checkstyle plugins.
Issues, Majority	Common Code Quality Issues	The most common code quality issues that the respondents mentioned are syntax errors, logic errors and runtime errors.
Challenges/Limitations	Research gap	Almost all the participants agreed that code generation tools have multiple code quality issues. Also, they stated that generated code had different types of code quality issues compared to generic syntax, logic and runtime errors. Hence, they thought that the current static analysis tools could use an update to adapt to the new issues being raised.
Detection, Satisfaction	Satisfaction	Majority of the respondents are mostly satisfied with the current tools as they detect most issues. However, respondents believe that they need to be updated to cater to generated code for the future.
Techniques	Trending techniques and room for improvement	Most respondents mentioned that AI tools are becoming very useful

		in detecting code related issues. There is a huge room for improvement and potential in using deep learning in detecting code quality issues.

Table 5: Thematic Analysis for Interviews

2.5.3 Summary of Findings

Findings	**LR**	**Interviews**	**Brainstorming**
AI-generated code especially commonly contains quality issues such as syntax, runtime and logic errors.	✓	✓	
Code quality issues are prevalent with the widespread adoption of AI-Generated code.	✓		✓
Lack of studies which formally investigate the reliability and quality of generated code.	✓	✓	
Code quality issues is a vital problem and existing tools need improvement.	✓	✓	
There is almost no work to detect code quality issues using deep learning methods.	✓		
Evaluation metrics such as TP, FP, recall precision can be used to evaluate the model	✓	✓	✓

Table 6: Summary of Findings

2.6 Context Diagram

A context diagram also known as a level 0 data flow diagram, is used to grasp the requirements and demonstration of how a system interacts to its related surroundings.

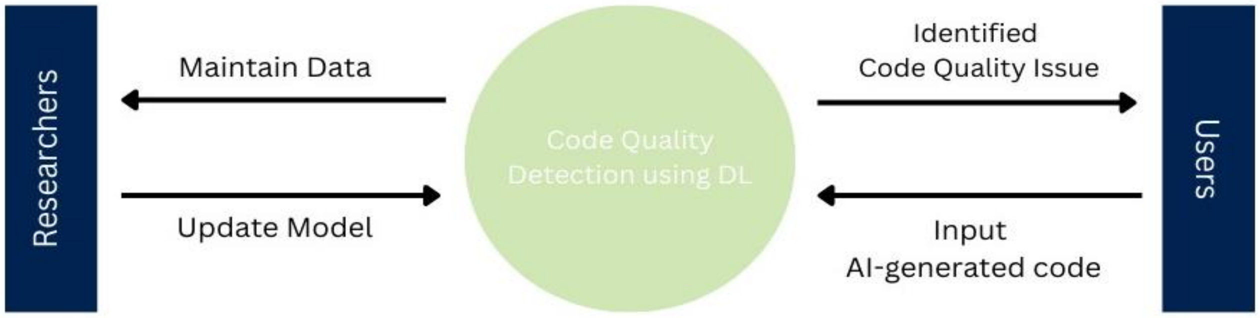


Figure 3: Context Diagram (self-composed)

2.7 Use Case Diagram

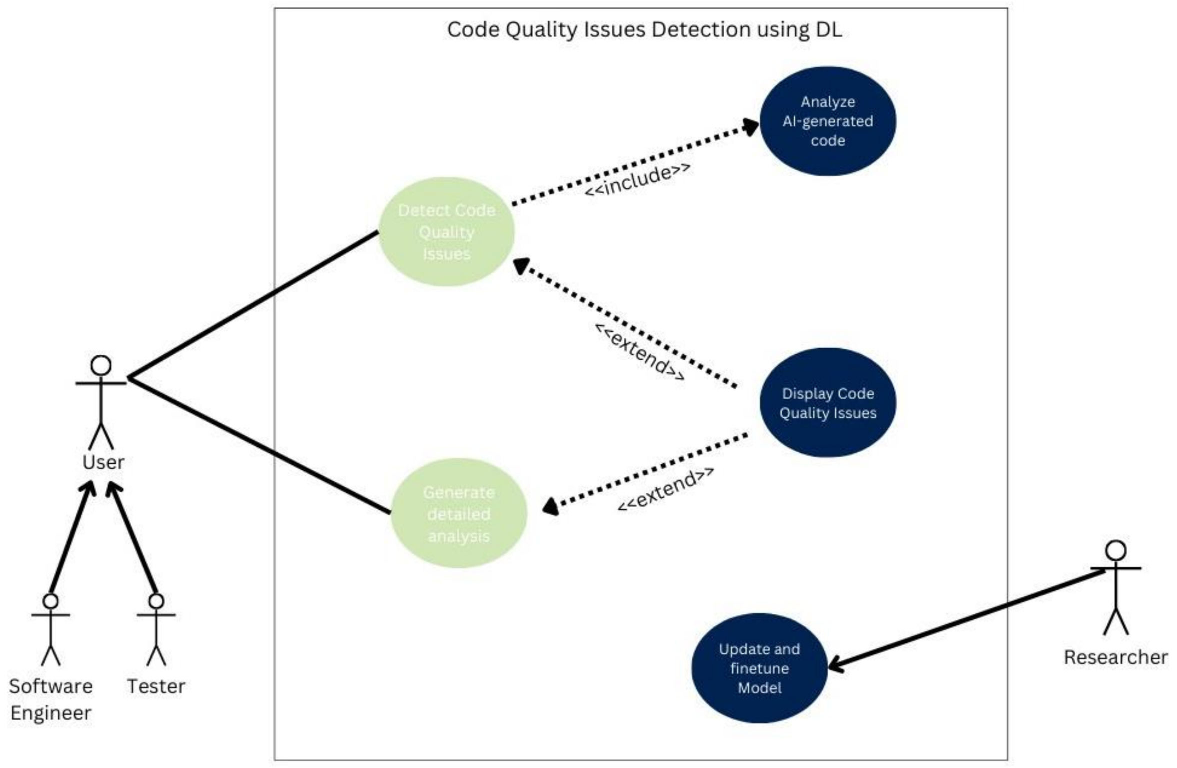

Code Quality Issues Detection using DL
Analyze AI-generated code
<<include>>
Detect Code Quality Issues
<<extend>>
Display Code Quality Issues
<<extend>>
Generate detailed analysis
User
Software Engineer
Tester
Update and finetune Model
Researcher

Figure 4: Use Case Diagram (self-composed)

2.8 Use Case Descriptions

Use Case	Detect Code Quality Issues
ID	UC01
Description	Detect code quality issues from provided code by user.
Primary actor	User
Secondary actor	None
Preconditions	The user must input the AI-generated code within the character limit and in supported language.
Main flow	**MF1** – User opens the application. **MF2-** User inputs AI-generated code. **MF3-** The system runs an analysis based on the input. **MF4-** Display output highlighting code quality issues in the AI-generated code given.
Alternative flow	None
Exceptional flow	**EF1** – User did not input code – code quality analysis won't be executed. **EF2-** User exceeded character limit – display error message. **EF3-** Language not supported – display error message.
Extended use case	Display Results.
Included use case	Analyze the code quality issues in the AI-generated code.
Postconditions	User is displayed a result with the detected code quality issues in the AI-generated code.

Table 7: Use Case 1 Description

2.9 Requirements

MoSCoW method was used to manage and determine the priority levels of the identified functional and non-functional requirements.

Priority Level	Description
Must have (M)	Requirements that are core and essential to develop a successful system.
Should have (S)	Requirements that could be considered important but not a necessity to develop the prototype.
Could have (C)	Requirements that are desirable and voluntary to implement.
Will not have (W)	Requirements that are not part of the scope for the system.

Table 8: MoSCoW Prioritization Levels

2.9.1 Functional Requirements

FR ID	Requirement	Priority Level
FR01	The system must be able to identify code quality issues in the inputted AI-generated code.	M
FR02	The system must be able to detect multiple code quality issues.	M
FR03	Users must be allowed to paste the AI-generated code in a commonly used programming language.	M
FR04	The GUI should display the type of code quality issues in a meaningful way.	S
FR05	The GUI should allow user to use different versions of the model for comparison.	C
FR06	The system could have validation checks to handle user errors and display specific error messages (ex: missing input, runtime errors)	C
FR07	The GUI could highlight where the code quality issue is being	C

	thrown and view a detailed description.	
FR08	User could be allowed to download a detailed report of the analysis performed and get the results in a PDF or a known file format.	C
FR09	User could get information about the accuracy and other evaluation metrics of the performed prediction	C
FR10	The system will fix all the code quality issues in the inputted AI-generated code automatically.	W

Table 9: Functional Requirements

2.9.2 Non-functional Requirements

NFR ID	**Requirement**	**Description**	**Priority Level**
NFR01	Reliability	The system should be reliable and accurate when detecting code quality issues in AI-generated code.	M
NFR02	Usability	The system should be effective in a way that the user can get information about the code quality issues in the inputted code.	M
NFR03	Performance	The system should not take too much time to generate the detected code quality issues.	S
NFR04	Scalability	The system must be deployed to cloud and handle large lengths of input data.	C
NFR05	Compatibility	The system must be compatible on multiple platforms and support multiple programming languages.	C
NFR06	Maintainability	The system must be well documented in case of future references and improvements.	C
NFR07	Security	The system must have defense mechanism to counter harmful and illegal attacks or requests.	W

Table 10: Non-functional Requirements

2.10 Chapter Summary

This chapter a rich picture diagram and a stakeholder onion model was presented to help identify the stakeholders. The author arranged all key users interacting with the system and showed how interactions would be maintained. Moreover, requirement elicitation methods and their rationales as well as respective conclusions were given. The use cases combined with descriptions and the functional and non-functional requirements were discussed towards the latter part of the chapter.

CHAPTER 03: DESIGN

3.1 Chapter Overview

In this chapter, the author focusses on the design of the system according to the design specifications and goals. A high-level diagram for the architecture will be shown to get an overview of the system alongside low-level design to dive into the architecture further. The desired UI, wireframes and further design diagrams and justifications will also be covered.

3.2 Design Goals

Design Goal	**Description**
Reliability	The output of the system should be accurate and reliable in identifying code quality issues in AI-generated code.
Usability	According to the requirement gathering period, it would be essential that new users can efficiently use the system without overcomplicating the input. Therefore, the system should be user friendly and efficient to use.
Performance	The system should ensure it does not take too long to perform the analysis and display the result
Maintainability	The goal should be that developers can use the proposed system in their everyday practices and researchers need to be able to build on top of the system. A well-maintained structure must be followed and documented.
Extendibility	The system should follow best software practices and structure to make sure expanding the system to support multiple languages would not cause major problems.

Table 11: Design Goals

3.3 High level Design

3.3.1 Architecture Diagram

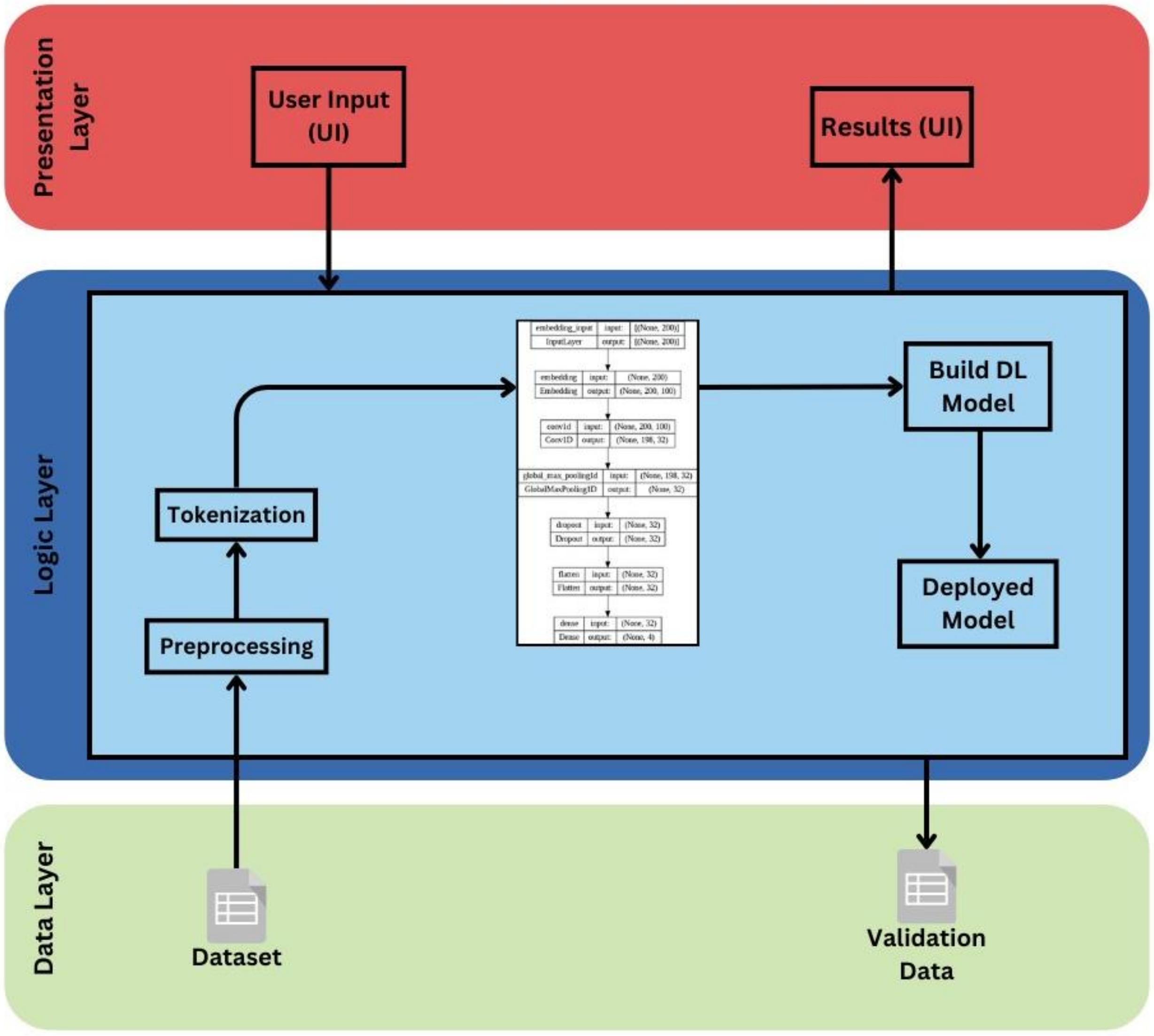

Presentation Layer
User Input (UI)
Results (UI)
Logic Layer
embedding_input | InputLayer | input: [(None, 200)] | output: [(None, 200)]
embedding | Embedding | input: (None, 200) | output: (None, 200, 100)
conv1d | Conv1D | input: (None, 200, 100) | output: (None, 198, 32)
global_max_pooling1d | GlobalMaxPooling1D | input: (None, 198, 32) | output: (None, 32)
dropout | Dropout | input: (None, 32) | output: (None, 32)
flatten | Flatten | input: (None, 32) | output: (None, 32)
dense | Dense | input: (None, 32) | output: (None, 4)
Build DL Model
Tokenization
Deployed Model
Preprocessing
Data Layer
Dataset
Validation Data

Figure 5: Layered Architecture Diagram (self-composed)

3.3.2 Discussion of layers of the Architecture

Presentation Layer

The interface that the user interacts with is illustrated in this layer. The input field which is connected to the logic layer which carries out the code quality issues detection and the results component which presents the findings.

Logic Layer

This layer is where the processing and core implementation happens. This layer contains pre processing so that the data is transformed into a format that can be used as input for a machine learning model and includes tokenization, layering and model building. It is connected to the UI layer to get both inputs and display the result to the user

Data Layer

This consists of the dataset and the validation dataset. The model uses the dataset to train and therefore its connected to the logic layer as it provides the training data for the model to preprocess. After the training of the data, the validation dataset is used to validate the model.

3.4 System Design

3.4.1 Choice of Design Paradigm

Object Oriented Analysis and Design (OOAD) and Structured Systems Analysis and Design Method (SSADM) are two commonly used design paradigms in software development. This project is inclined to alterations towards betterment along with the latest research as it is a new field of research. Accordingly, SSADM is regarded as the optimum selection for the present work. So, using the **SSADM** approach due to its precision and simplicity as it's a growing system would be the most optimal.

3.5 Detailed Design Diagrams

3.5.1 Data Flow Diagram

The level 1 DFD (Data Flow Diagram) has been illustrated below which shows a clear presentation of the broken-down components of the system and the relationships and dataflow between them.

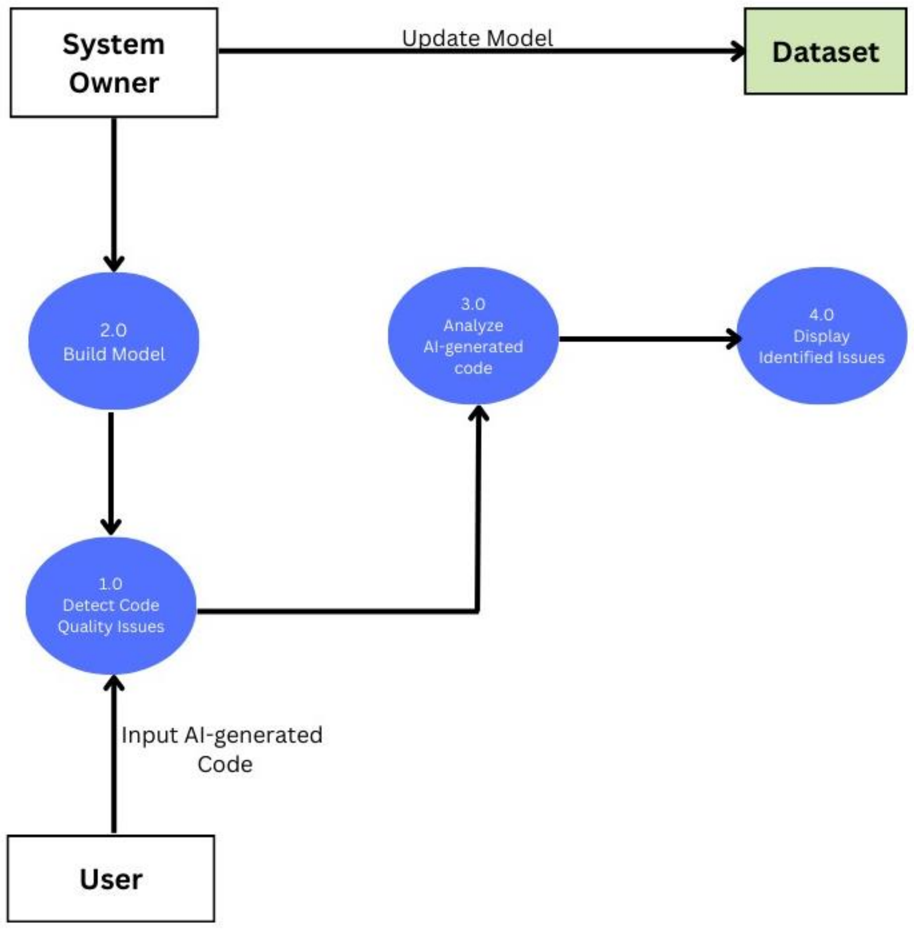

System Owner
Update Model
Dataset
2.0 Build Model
3.0 Analyze AI-generated code
4.0 Display Identified Issues
1.0 Detect Code Quality Issues
Input AI-generated Code
User

Figure 6: Data Flow Diagram - Level 1 (self-composed)

A level 2 DFD (Data Flow Diagram) is illustrated below. It shows a deeper understanding of the system compared to a level 1 DFD. It breaks down the components of the system further into subcomponents and processes showing the data flow between them.

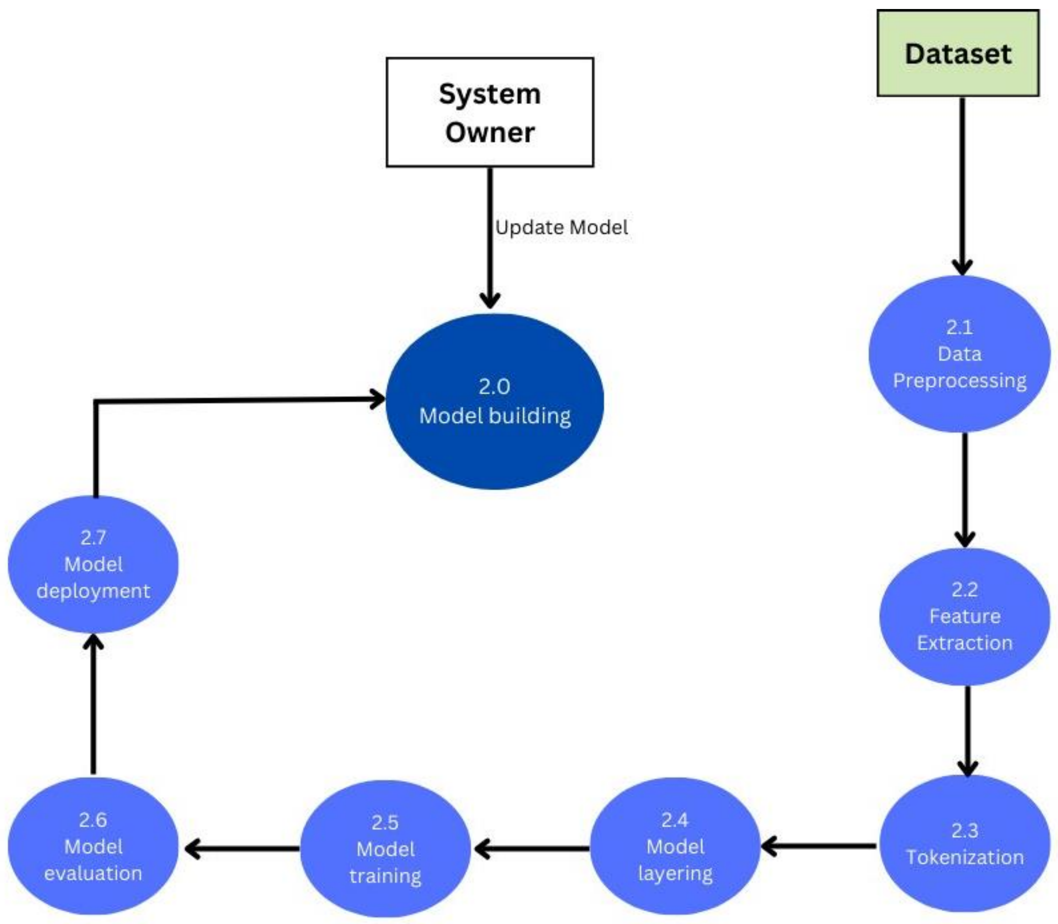


Figure 7: Data Flow Diagram - Level 2 (self-composed)

3.5.2 System Process Flow Chart

The below illustrated diagram shows the flow of the proposed system. It also shows the order of steps taken to get the desired outcome. This gives a clear visualization to understand the process from start to end in a system.

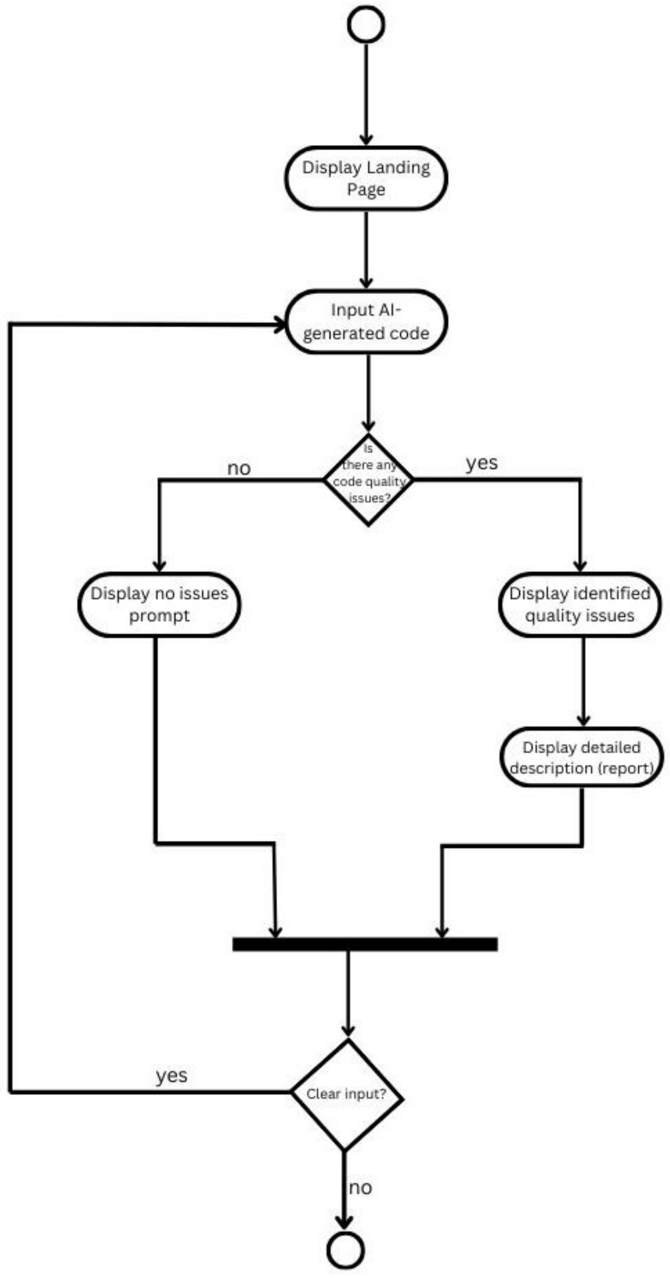
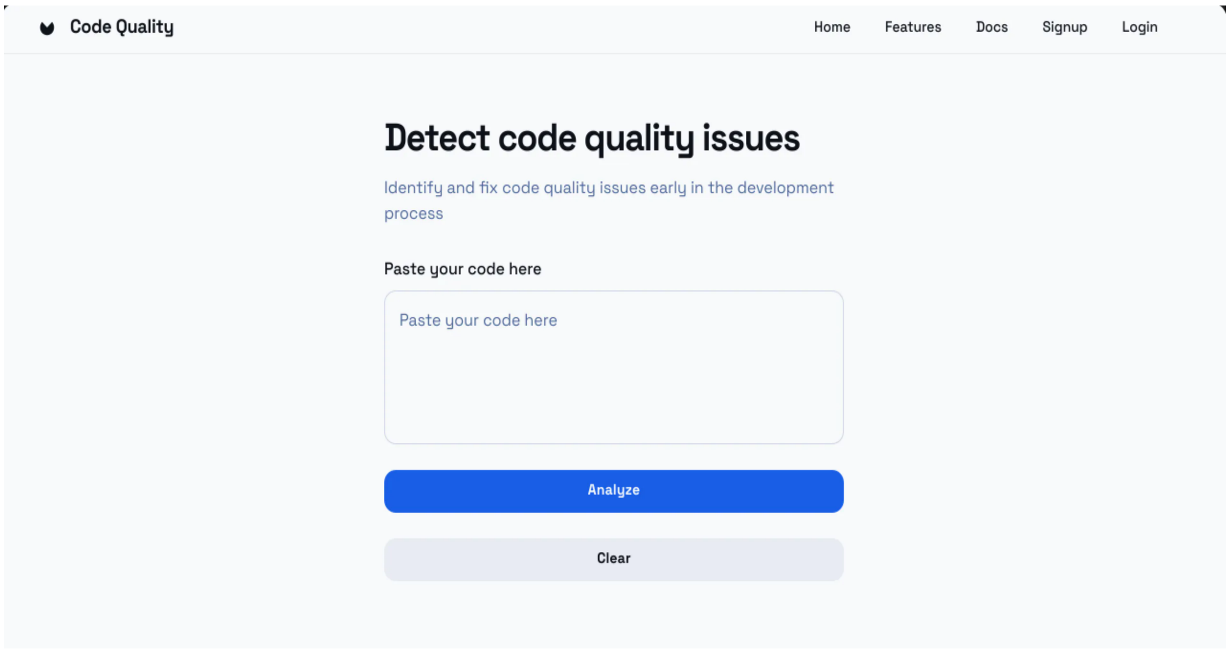

Display Landing Page
Input AI-generated code
Is there any code quality issues?
no
yes
Display no issues prompt
Display identified quality issues
Display detailed description (report)
Clear input?
yes
no

Figure 8: Activity Diagram (self-composed)

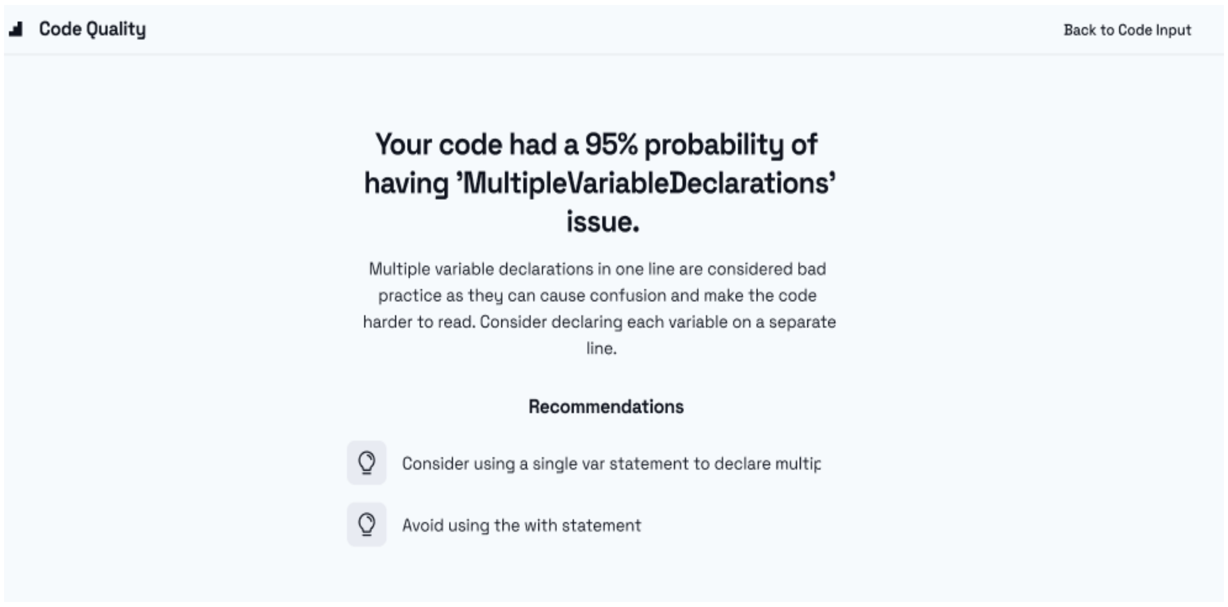
3.5.3 User Interface Design



Detect code quality issues

Identify and fix code quality issues early in the development process

Figure 9: UI for Input Page (Main Page)



Your code had a 95% probability of having 'MultipleVariableDeclarations' issue.

Multiple variable declarations in one line are considered bad practice as they can cause confusion and make the code harder to read. Consider declaring each variable on a separate line.

Consider using a single var statement to declare multip

Avoid using the with statement

Figure 10: UI for Output Page (Displaying Results)

3.6 Chapter Summary

The conceptualized system design, goals and associated approaches were deeply explained and illustrated in this chapter for the system. The related design diagrams which have been catered to organize and depict high and low-level architectural components according to the design objectives have been illustrated. After that, the selected pattern of software design has been well described. The diagrams that support each architecture component, data flow and user interface were explored.

Chapter 04: INITIAL IMPLEMENTATION

4.1 Chapter Overview

This chapter provides the initial implementation for the proposed system. It also covers the tools used, programming languages, libraries and the overall technology stack along with their justifications. According to the design and requirement elicitation, the initial core functionalities of the system will be provided with code snippets to demonstrate the POC.

4.2 Technology Selection

4.2.1 Technology Stack

The figure below illustrates the tools and technologies selected for the implementation of the project. The technology for each layer has been presented.

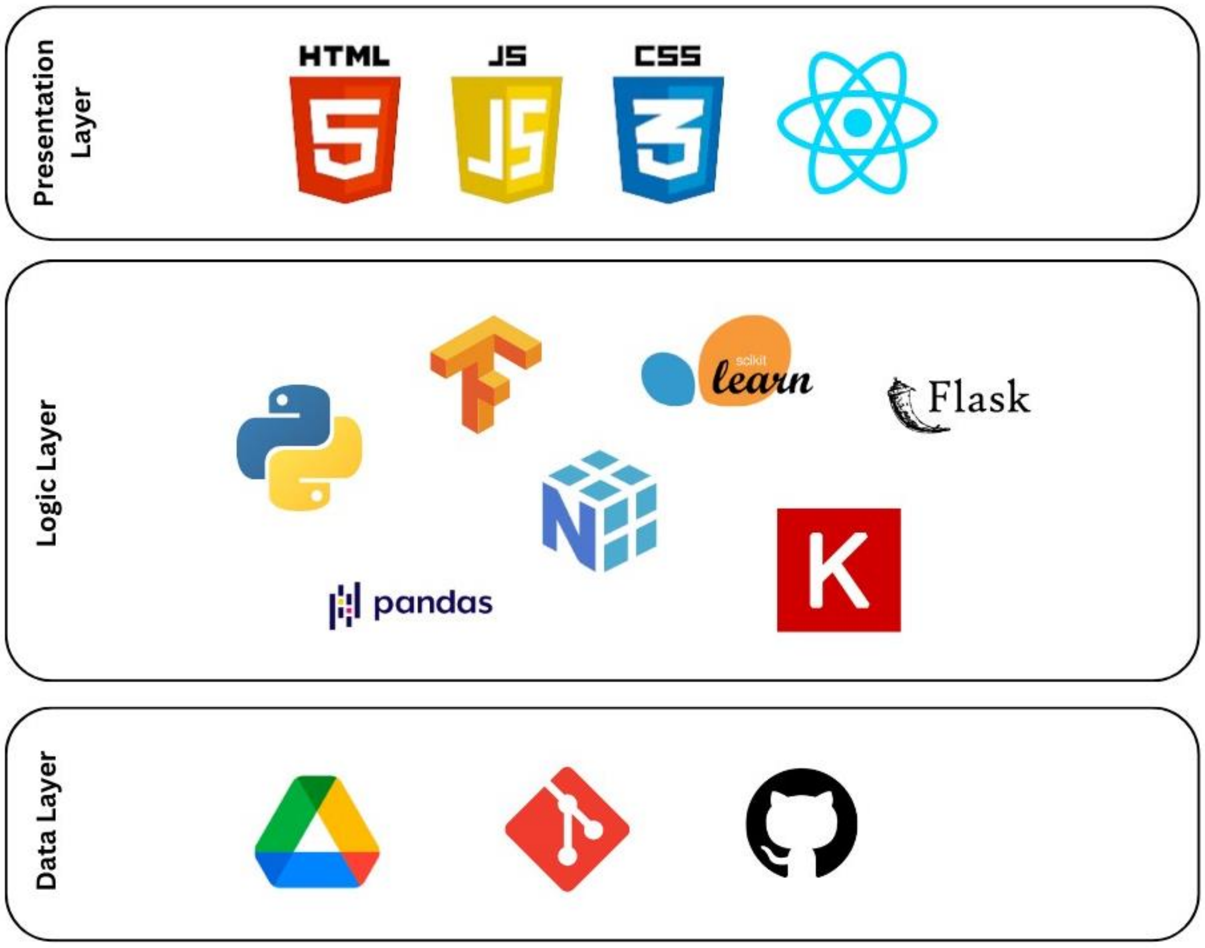


Figure 11: Technology Stack (self-composed)

4.2.2 Dataset Selection

The dataset used for this project contains 2000+ scenarios of leetcode questions and the attempted solution by AI. The fields contain the code quality information if any code quality issues exist in the generated code. It was recently published in a research paper by Monash University Australia and the dataset was published under the MIT license to support the open source initiative and continue the exploration of AI-generated code (Liu et al., 2023).

4.2.3 Development Frameworks

Development Framework	Justification
Tensorflow	Tensor flow is an updated and widely used framework for building ML and DL models. It is highly flexible, scalable and efficient for the purpose of this project.
Flask	Flasks' lightweight nature is easy to learn and allows to flexibility. The ease of learning makes it a popular framework to build API's and for integration purposes.

Table 12: Framework Selection Justifications

4.2.4 Programming Languages

Programming Language	Justification
Python	Python was selected due to the updated libraries and frameworks that allows to seamlessly build ML and DL models. It is specialized and has a lot of community support and examples for the nature of this project.
JavaScript	JavaScript was selected as it is a recommended and ideal choice for developing responsive web pages It also has extensive library support and cross -platform compatibility.

Table 13: Programming Languages Justifications

4.2.5 Libraries

Library	Justification
Pandas	A powerful library that allows preprocessing such as cleaning, filtering and working with structured data using data frames efficiently and time effective.

NumPy	A powerful and widely used library for mathematical computations. Allows to manipulate arrays and work with multi-dimensional data structures. One of the most important libraries to build deep learning models.
Matplotlib	Allows to visualize and understand the data for detecting code quality issues. Supports a variety of visualization tools and techniques.
Scikit-learn	Efficient library in splitting data seamlessly and useful for feature scaling and evaluating a model’s performance.

Table 14: Libraries Selection Justifications

4.2.6 IDE

IDE	Justification
Google Colab	Allows access to powerful computational resources which may not be accessible for a local machine. Also, has pre installed libraries saving the time and effort to manually install each library locally. Efficient for the nature of this project and supports Python extensively.
VSCode	This IDE is user friendly and used widely by developers for front end and backend projects. The rich eco system of extensions and built in git integration allows developers to seamlessly write code efficiently.

Table 15: IDE Selection Justifications

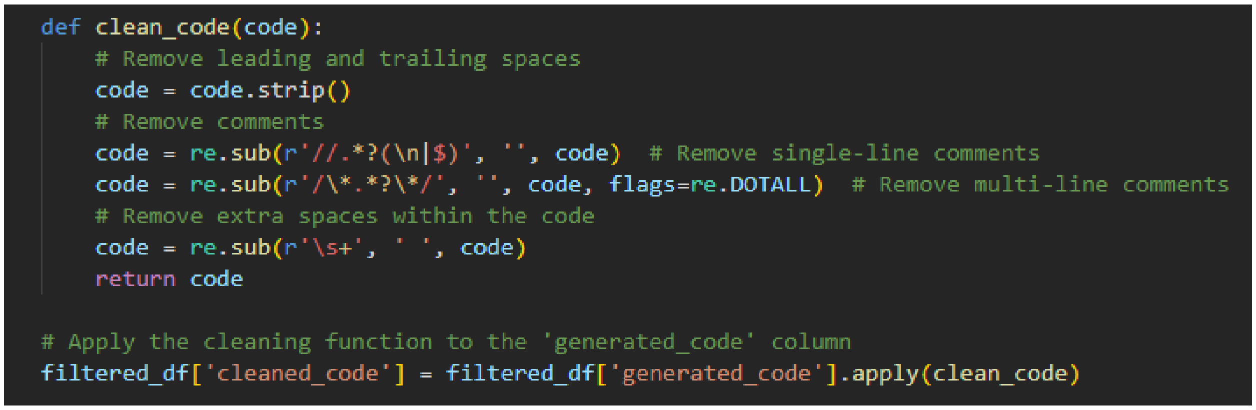
4.2.7 Summary of Technology Selection

Component	Tools
Development Frameworks	Tensorflow, Flask
Programming Languages	Python, JavaScript
Libraries	Pandas, Numpy, Matplotlib, Scikit-learn
IDE	Google Colab, VSCode
Version Control	Git, GitHub

Table 16: Summary of Technology Selection

4.3 Implementation of the Core Functionality

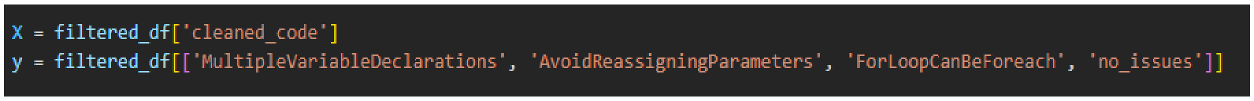
The author initially had to preprocess the data by cleaning the code which involved removing leading and trailing spaces from code, remove comments (single-line and multi-line) and remove unnecessary extra spaces within the code. This is to ensure that the model is not influenced by irrelevant formatting details and focus more on the code itself and its logic.



```python
def clean_code(code):
    # Remove leading and trailing spaces
    code = code.strip()
    # Remove comments
    code = re.sub(r'//.*?(\n|$)', '', code)  # Remove single-line comments
    code = re.sub(r'/\*.*?\*/', '', code, flags=re.DOTALL)  # Remove multi-line comments
    # Remove extra spaces within the code
    code = re.sub(r'\s+', ' ', code)
    return code

# Apply the cleaning function to the 'generated_code' column
filtered_df['cleaned_code'] = filtered_df['generated_code'].apply(clean_code)
```

Figure 12: Data Cleaning

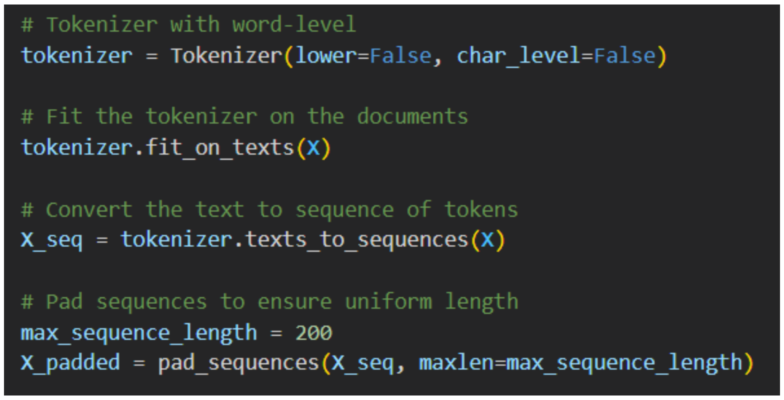
Then the input and target variables are defined. In this case, in our dataset we have a ‘generated_code’ column which contains AI-generated code. The target variables were extracted using the data extraction phase and can be visible in the repository link shared below.



```python
X = filtered_df['cleaned_code']
y = filtered_df[['MultipleVariableDeclarations', 'AvoidReassigningParameters', 'ForLoopCanBeForeach', 'no_issues']]
```

Figure 13: Defining Input and Target Variables

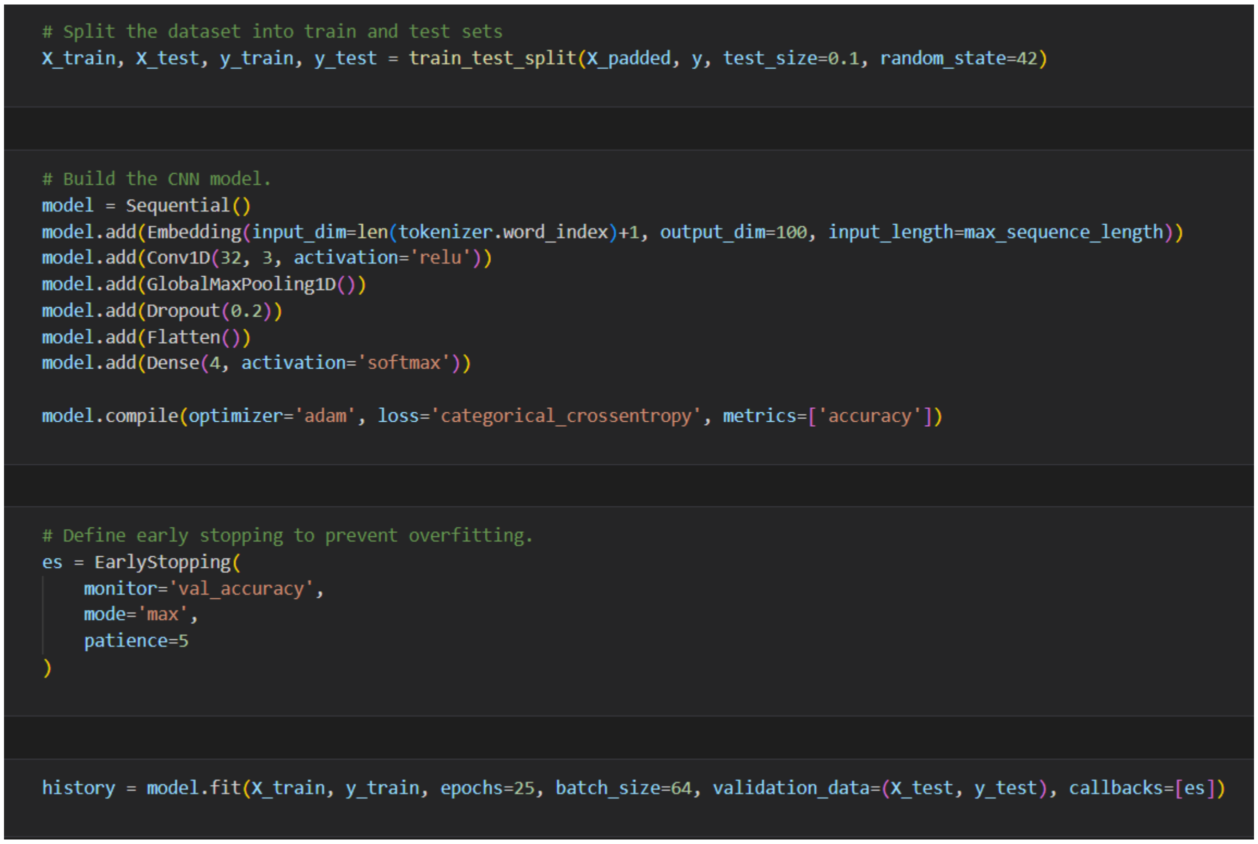
Tokenization was done because the input require consistent shape and length. This Python code is used to preprocess text data for use in a machine learning model. By converting the text into sequences of integers and padding these sequences, the text data is transformed into a format that can be used as input for a machine learning model.



```
# Tokenizer with word-level
tokenizer = Tokenizer(lower=False, char_level=False)

# Fit the tokenizer on the documents
tokenizer.fit_on_texts(X)

# Convert the text to sequence of tokens
X_seq = tokenizer.texts_to_sequences(X)

# Pad sequences to ensure uniform length
max_sequence_length = 200
X_padded = pad_sequences(X_seq, maxlen=max_sequence_length)
```

Figure 14: Tokenization

The next step was to split the data into train and test sets and feed into the model. For this POC, a CNN model was created with an embedding layer, a convolution layer, a max pooling layer, a dropout layer, a flatten layer and finally an output layer. Callbacks such as early stopping were defined to counter overfitting and the model was trained with 25 epochs and batch size 64.



```
# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.1, random_state=42)
```

```
# Build the CNN model.
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=100, input_length=max_sequence_length))
model.add(Conv1D(32, 3, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(4, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Define early stopping to prevent overfitting.
es = EarlyStopping(
    monitor='val_accuracy',
    mode='max',
    patience=5
)
```

```
history = model.fit(X_train, y_train, epochs=25, batch_size=64, validation_data=(X_test, y_test), callbacks=[es])
```

Figure 15: Building and Training Model

The entire implementation file will be available on GitHub on a private repository and access will be given if requested.

4.4 Chapter Summary

This chapter covered the main areas of the initial implementation. The anticipated technology stack, libraries, IDE's, frameworks and the overall technology selection along with their justifications were mentioned. Code snippets were provided which covered the core implementation and functionalities of the initially proposed prototype. The following chapter will cover the areas of initial testing results and evaluation, it will further highlight the necessary steps required to further improve the POC to ensure a remarkable MVP.

CHAPTER 05: CONCLUSION

5.1 Chapter Overview

This chapter concludes the PSDP and highlights the initial test results. It further explores any deviations made to the project based on previously set timelines. Improvements that are required and essential will be covered towards the end of the chapter along with a demo video of the prototype explaining the problem, solution and improvements. A link to access the codebase will also be provided.

5.2 Deviations

5.2.1 Scope Related Deviations

According to the scopes mentioned in the project proposal, there have been no deviations from both in-scope and out-scope categories. There was no reason to add or remove any additional functionalities as the in scope covers the problem and solution novelty effectively.

5.2.2 Schedule Related Deviations

There are no schedule related deviations as of February 2024, the below tables shows the expected tasks of this project along with their status and deadlines.

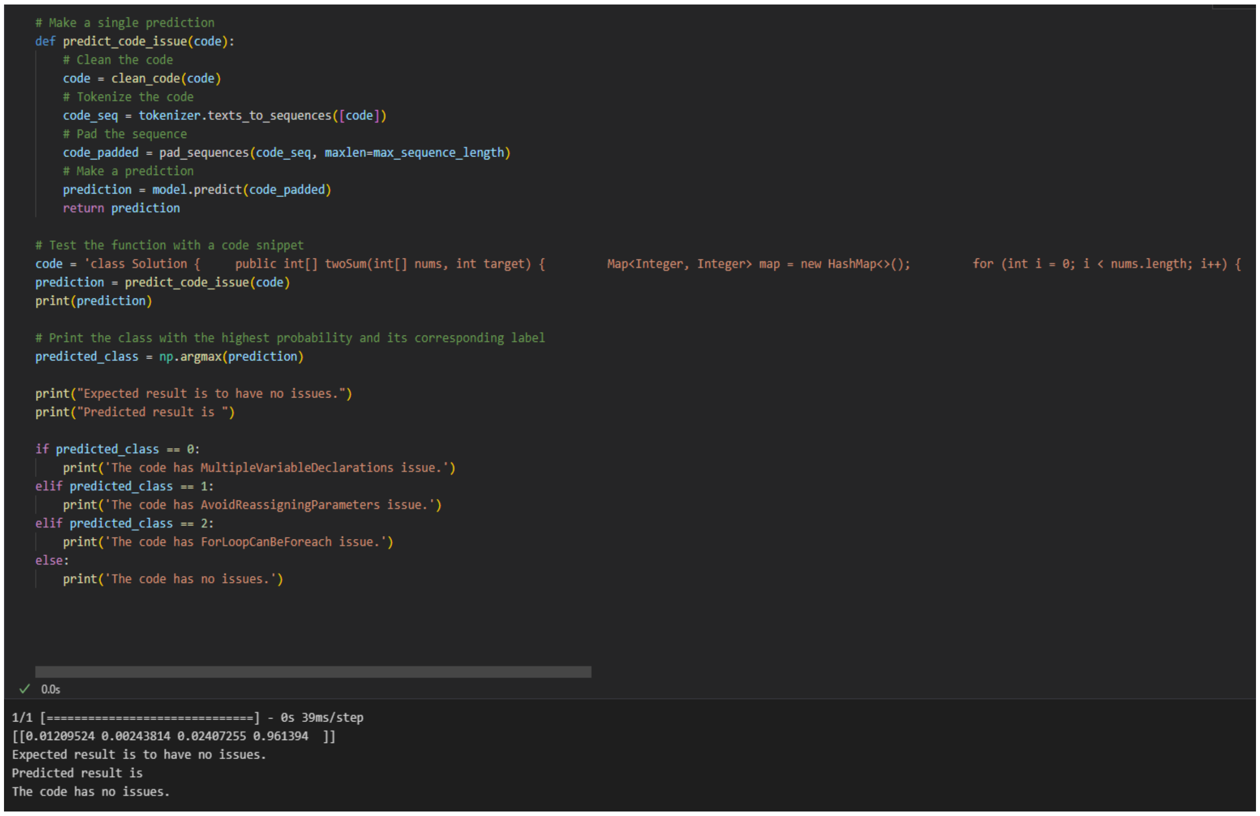
Deliverable	**Deadline**	**Status**
Final Project Proposal	October 2023	Completed
Initial Literature Review Thorough analysis and critical evaluation of existing and relatedwork in the domain.	October 2023	Completed
System Requirement Specification Specifies the requirements to be met by the proposed prototype.	November 2023	Completed

Final Project Specifications Design and Prototype (PSDP) Documentation and POC of the implemented solution alongwith its core features.	February 2024	Completed
Minimum Viable Product Submission	March 2024	In progress
Testing and Evaluation	March 2024	In progress
Final Thesis Submission	April 2024	In progress

Table 17: Schedule Related Deviations

5.3 Initial Test Results

The expected use case for how the implementation will be used is shown below where the expected results for the model to predict that there are no issues in the code.



```
# Make a single prediction
def predict_code_issue(code):
    # Clean the code
    code = clean_code(code)
    # Tokenize the code
    code_seq = tokenizer.texts_to_sequences([code])
    # Pad the sequence
    code_padded = pad_sequences(code_seq, maxlen=max_sequence_length)
    # Make a prediction
    prediction = model.predict(code_padded)
    return prediction

# Test the function with a code snippet
code = 'class Solution {     public int[] twoSum(int[] nums, int target) {         Map<Integer, Integer> map = new HashMap<>();         for (int i = 0; i < nums.length; i++) {
prediction = predict_code_issue(code)
print(prediction)

# Print the class with the highest probability and its corresponding label
predicted_class = np.argmax(prediction)

print("Expected result is to have no issues.")
print("Predicted result is ")

if predicted_class == 0:
    print('The code has MultipleVariableDeclarations issue.')
elif predicted_class == 1:
    print('The code has AvoidReassigningParameters issue.')
elif predicted_class == 2:
    print('The code has ForLoopCanBeForeach issue.')
else:
    print('The code has no issues.')
```

```
1/1 [==============================] - 0s 39ms/step
[[0.01209524 0.00243814 0.02407255 0.961394  ]]
Expected result is to have no issues.
Predicted result is
The code has no issues.
```

Figure 16: Making a Prediction

The POC accurately identifies the expected outcome and highlights the probabilities that the code has the code quality issues namely 'MultipleVariableDeclarations', 'AvoidReassigningParameters' and 'ForLoopCanBeForeach'. Therefore, for a POC it could be said that this is the correct approach and intended use case.

5.3.1 Confusion Matrix

Since the prototype is a muti-class classification, a confusion matrix would be ideal to test the false positives and understand the misconceptions further.

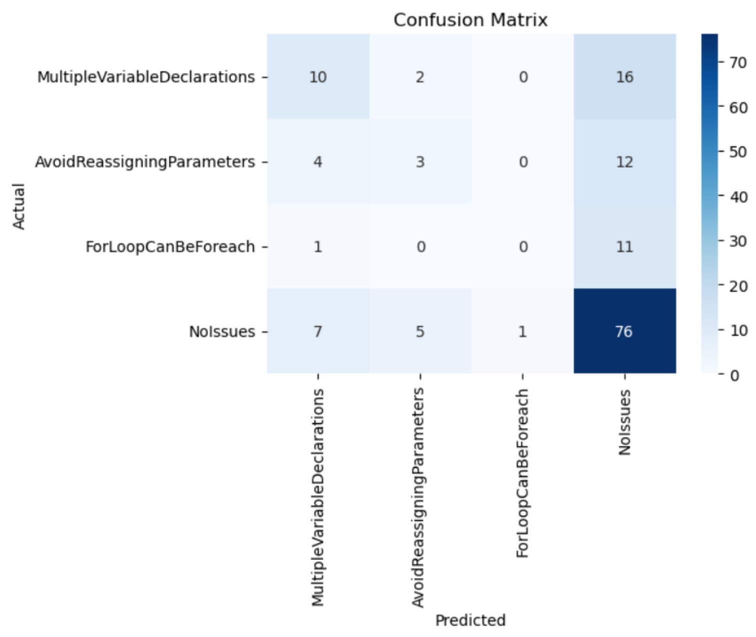


Figure 17: Confusion Matrix

From the confusion matrix it could be understood that the data imbalance problem is causing more false positives especially in the case of incorrectly predicting no issues. The author aims to address this problem and is further discussed in the following section.

5.4 Required Improvements

Based on the initial results, the following improvements were identified as essential for the MVP which will be presented with the thesis in April. The author noticed a data imbalance problem alongside revisiting the architecture for the best possible output and accuracy to avoid false positives witnessed as shown previously.

1. The author aims to include weighted classes so that the model can pay attention to minority classes.

2. The author aims to incorporate sampling (resampling and oversampling) to feed more

data with the minority classes and avoid the model training massively on imbalanced data especially with no issues.

3. The author aims to revisit and improve the architecture of the CNN and experiment further with hyperparameter tuning to achieve the best possible architecture and layers for the CNN.

5.5 Demo of the Prototype

Below link includes the demo video of the prototype explaining the problem, solution and improvements.

- Demo Video

Below link redirects to the GitHub repository of the current implementation. Access to the repository will be provided when requested.

- GitHub

5.6 Chapter Summary

This chapter discussed the initial test results. It further explored any deviations made to the project based on previously set timelines. Improvements that are required which are essential were covered which the author aims to address and ensure the best novelty prototype. Finally, the link to the demo video of the prototype explaining the problem, solution and improvements and the codebase was provided.

References

Alzubaidi, L. et al. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8 (1), 53. Available from https://doi.org/10.1186/s40537-021-00444-8.

Barke, S., James, M.B. and Polikarpova, N. (2023). Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages*, 7 (OOPSLA1), 85–111. Available from https://doi.org/10.1145/3586030.

Birillo, A. et al. (2023). Detecting Code Quality Issues in Pre-written Templates of Programming Tasks in Online Courses. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 152–158. Available from https://doi.org/10.1145/3587102.3588800.

Das, A.K., Yadav, S. and Dhal, S. (2019). Detecting Code Smells using Deep Learning. *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*. October 2019. Kochi, India: IEEE, 2081–2086. Available from https://doi.org/10.1109/TENCON.2019.8929628 [Accessed 19 October 2023].

Harding, J. et al. (2023). AI language models cannot replace human research participants. *AI & SOCIETY*. Available from https://doi.org/10.1007/s00146-023-01725-x [Accessed 3 October 2023].

Ho, A. et al. (2023). Fusion of deep convolutional and LSTM recurrent neural networks for automated detection of code smells. *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 14 June 2023. Oulu Finland: ACM, 229–234. Available from https://doi.org/10.1145/3593434.3593476 [Accessed 3 October 2023].

Juddoo, S. and George, C. (2020). A Qualitative Assessment of Machine Learning Support for Detecting Data Completeness and Accuracy Issues to Improve Data Analytics in Big Data for the Healthcare Industry. *2020 3rd International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering (ELECOM)*. 25 November 2020. Balaclava, Mauritius: IEEE, 58–66. Available from https://doi.org/10.1109/ELECOM49001.2020.9297009 [Accessed 19 October 2023].

Kanan, A., Sherief, N. and Abdelmoez, W. (2022). An Intelligent Framework based on CNN and Neutrosophic Technique for Detecting Code Smells and Ranking Code. *2022 32nd International Conference on Computer Theory and Applications (ICCTA)*. 17 December 2022. Alexandria, Egypt: IEEE, 31–37. Available from https://doi.org/10.1109/ICCTA58027.2022.10206227 [Accessed 10 February 2024].

Liu, Y. et al. (2023). Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *ArXiv*. Available from https://www.semanticscholar.org/paper/Refining-ChatGPT-Generated-Code%3A-Characterizing-and-Liu-Le-Cong/41a2e7c079179ae94557d3198de674a16a5987a6 [Accessed 31 August 2023].

Mamun, A. et al. (2019). Improving Code Smell Predictions in Continuous Integration by Differentiating Organic from Cumulative Measures. 2019. Available from https://www.semanticscholar.org/paper/Improving-Code-Smell-Predictions-in-Continuous-by-Mamun-Staron/db3902988f16e62a69a0b4afa5f77c0d63afd1fe [Accessed 14 February 2024].

Mouselinos, S., Malinowski, M. and Michalewski, H. (2023). A Simple, Yet Effective Approach to Finding Biases in Code Generation. Available from http://arxiv.org/abs/2211.00609 [Accessed 21 September 2023].

Pecorelli, F. et al. (2019). Comparing Heuristic and Machine Learning Approaches for Metric-Based Code Smell Detection. *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. May 2019. Montreal, QC, Canada: IEEE, 93–104. Available from https://doi.org/10.1109/ICPC.2019.00023 [Accessed 13 November 2023].

Raychev, V. (2021). Learning to Find Bugs and Code Quality Problems - What Worked and What not? *2021 International Conference on Code Quality (ICCQ)*. 27 March 2021. Moscow, Russia: IEEE, 1–5. Available from https://doi.org/10.1109/ICCQ51190.2021.9392977 [Accessed 19 October 2023].

Sengamedu, S. and Zhao, H. (2022). Neural language models for code quality identification. *Proceedings of the 6th International Workshop on Machine Learning Techniques for Software Quality Evaluation*. 7 November 2022. Singapore Singapore: ACM, 5–10. Available from https://doi.org/10.1145/3549034.3561175 [Accessed 5 February 2024].

Sharma, K.K., Sinha, A. and Sharma, A. (2022). Software Defect Prediction using Deep

Learning by Correlation Clustering of Testing Metrics. *International Journal of Electrical and Computer Engineering Systems*, 13 (10), 953–960. Available from https://doi.org/10.32985/ijeces.13.10.15.

Vable, A.M., Diehl, S.F. and Glymour, M.M. (2021). Code Review as a Simple Trick to Enhance Reproducibility, Accelerate Learning, and Improve the Quality of Your Team's Research. *American Journal of Epidemiology*, 190 (10), 2172–2177. Available from https://doi.org/10.1093/aje/kwab092.

Wu, J.J. (2023). Does Asking Clarifying Questions Increases Confidence in Generated Code? On the Communication Skills of Large Language Models. Available from http://arxiv.org/abs/2308.13507 [Accessed 4 September 2023].

Yetiştiren, B. et al. (2023). Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. Available from https://doi.org/10.48550/ARXIV.2304.10778 [Accessed 5 February 2024].

Appendix A – SRS

Interview Questions

Question	**Purpose** i.e. how it is linked to the research?	**Research Question**
What code analysis tools do you currently use?	To identify widely used code analysis tools in the industry.	RQ2
If you use any, does it accurately detect code quality issues?	To evaluate the need for a more accurate tool.	RQ1
What are the challenges or limitations of using existing tools or methods to detect code quality issues in AI-generated code?	To identify current gaps and limitations in current tools.	RQ1
What are the common code quality issues that are detected from the current tool you use?	To identify the common code quality issues that needs to be covered in the prototype.	RQ1
How satisfied are you with the current code analysis tools you use?	To understand if the developers are satisfied with the current code analysis tools they use.	RQ2
Do you know any trends or systems that are using or planning to use deep learning for code analysis?	To explore existing work and potential trends.	RQ3

Interviewee List

Interviewee ID	Name	Designation
1	Mr Mohamed Fazal	Software Engineer/Full Stack Developer, Tecciance (Pvt) Ltd.
2	Mr Ihthisam Rasheed	Senior QA Innovative-e (Pvt) Ltd
3	Mr Ayyoob Rifdhi	Senior Systems Engineer WSO2
4	Mr Farwaiz Firthouse	Intern Software Engineer, Innovative-e (Pvt) Ltd
5	Anonymous	QA Intern, WSO2