INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

QualityCheck AI: Detecting Code Quality Issues in AI-generated code using deep learning

A dissertation 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: April 2024

Abstract

The current tech era is rapidly evolving due to generative AI. The increase in AI- generated code in the industry raises 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, recent research has demonstrated effectiveness in using deep learning techniques to detect code smells and software vulnerabilities accurately. So the author explores a novel approach to design and execute a CNN trained on AI-generated code. This project would be the first in its nature to apply deep learning in identifying code quality issues particularly with AI-generated code.

The test results indicated that the proposed system successfully detected code quality issues providing accurate predictions with an accuracy of 95% and 94% for the code quality issues 'AvoidReassigningParameters' and 'ForLoopCanBeForEach' respectively. 82% and 80% accuracy were obtained for 'MultipleVariableDeclarations' and the no issues classification along with other well-performing evaluation metrics. This suggests that the proposed architecture achieves outstanding performance in detecting code quality issues in AI-generated code.

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

Declaration

I sincerely declare that this research project and everything associated alongside has not been previously submitted and is my own work. Any external information gathered have been given due accrediation to initial authors using citations.

Student Name: Areeb Niyas

Registraiton Number: w1809939/20200129

[illegible]

Signature: …………………..   Date: April 4th 2024

Acknowledgement

There has been a lot of time and effort put into this research project. With all ups and downs, you grow through what you go through. In this journey,I would like to express my gratitute to the ones who made this possible.

Firstly, “Alhamdulillah” translating to praise be to God in Arabic. Then I would like to thank my supervisor Mr Jihan Jeeth for his continous and unwavering support. Despite it being the midle of the day or night, he was always open to lend a helping hand and constantly checked up on the status of the project. So from the bottom of my heart, thank you.

Afterwards, I would like to thank my parents for their support and for always being there through thick and thin. They are the true heroes of your life who I’m blessed and forever grateful to have. I am who I am today because of you.

Moreover, I express my sincere thanks to the final year project module team and the lecturers for being a supporting pillar throughout my final year project. I would also like to extend my gratitude to the evaluators and interviewees who helped me gain invaluable feedback and a deeper understanding of the research domain.

Finally, I would like to thank my friends and extended family for being the go to people when everything may seem overwhelming. Thank you for always making me feel home wherever I go.

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
SCA	Static Code Analysis

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 methodology 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 ML algorithms are becoming popular in identifying code quality issues, recent studies that use DL methods have proved to be more promising. This includes the use CNNs, ANNs and RNNs which stand for convolutional neural network, artificial neural network and recurrent neural networks respectively. These algorithm have produced higher accuracy in detecting code smells better than their predecessors which are ML algorithms (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 techniques 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 process that provides a solution for the breakdown of above-mentioned problemsis required to ensure a novelty prototype. This study therefore would aim to create a tool that canbe 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 methods 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 DL 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 DL 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 perspective 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 foregather datasets that successfully produce results in the domain of using DL to detect code quality issues in code (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: LITERATURE REVIEW

2.1 Chapter Overview

This chapter aims to dive into the overall literature of the project and get thorough insights about the background and domain. Also, the existing work will be explored further to critically analyze its strengths and limitations gaining a deeper understanding of generative-AI and how it is related to code quality.

2.2 Concept Map

The concept map was maintained and illustrated throughout the LR. It helps visually present the path taken amongst the various possibilities for this project. It also covers the metrics used for evaluation and key technologies in this domain of research. It can be viewed in **APPENDIX A**.

2.3 Problem Domain

2.3.1 Rise of Generative AI

Generative AI has taken the world by storm and is being widely used in everyone's day to day lives. This is because it has exponentially evolved with the modern equipment and provides efficiency for all types of users (Lawlor and Chang, 2024).

The projected users for Generative AI is illustrated in the graph below which shows a rapid rise and implies an urgent need of adaption.

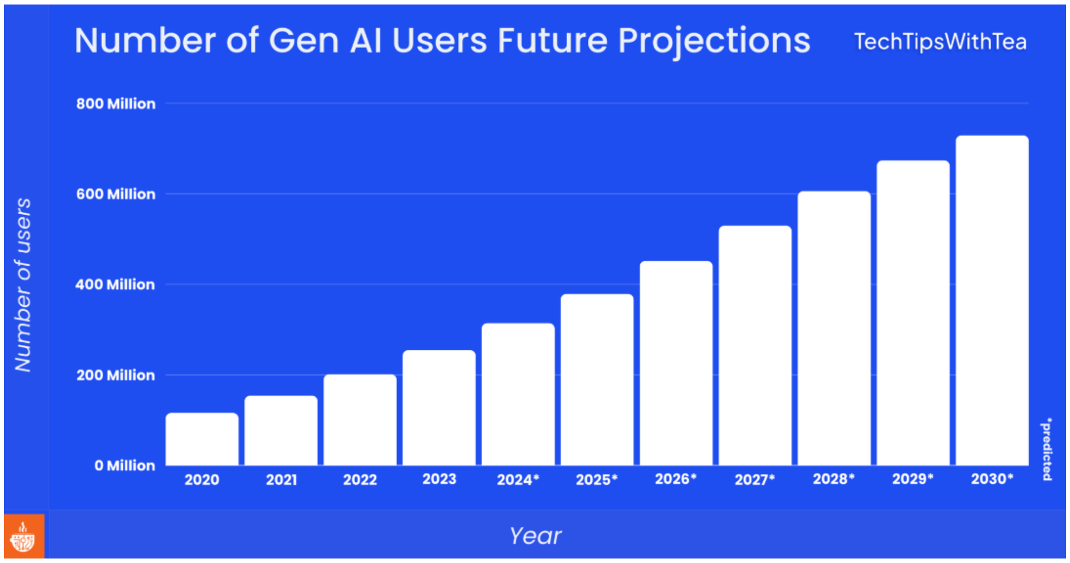


Figure 1: Usage statistics estimation for Generative AI (Singh, 2024)

Tools like ChatGPT that are generative AI helps people learn by centralizing the information and resources. Through the form of prompting, people across all age groups can educate themselves on various domains from school-level context to advanced topics that may not be accessible easily. The future is definitely heading towards AI-assisted lifestyles and all people are adapting rapidly. (Chang and Kidman, 2023). However, with the rapid adoption of tools like ChatGPT, rises potential risks and issues as it has been proved to have significant errors and challenges such as invalid answers, modified responses out of context are some among many (Murugesan and Cherukuri, 2023).

2.3.2 Challenges of Generative AI

Currently majority of users find that ChatGPT has the answer all, if not most of their problems including questions asked about their academics in their degrees and masters programs (Lock, 2022). It also takes it a step further by solving coding solutions and problems (Scharth, 2022). This in turn raises concerns about academic validity and integrity and the need to put academic policies in place to maintain this particular concern by working together with academic institutions to make it a responsible by addressing the current concerns (Eke, 2023). Some of the main challenges are discussed below:

1. **Privacy and ethics**

   Privacy involves the information of sensitive data which someone would not want it to be exposed and traditional approaches top solve these issues are not applicable when it comes to large volumes of data in LLMs (Fang et al., 2017). There has been incidents where ChatGPT conversations have been leaked due to data leakage incidents from ChatGPT which raises questions about the everyday operations and activities that some users find personal or important (Porter, 2023).

2. **Misuse**

   As discussed previously, academic integrity is often questioned and the concerns for students where they can cheat in examinations and write content using ChatGPT which is considered plagiarism all with an access to a tech device for a short period of time (Thorp, 2023). It also goes further than academics to gain knowledge and execute illicit activities because generative AI tools have a thin line in which is considered appropriate and inappropriate (Susarla et al., 2023).

3. **Transparency**

   Currently used algorithms and mechanisms in Generative AI tools are often considered as black boxes because the algorithms and internal workings are hidden (Fui-Hoon Nah et al., 2023). This non-explainability of systems that power AI tools threatens the validity and comprehensiveness of such tools because the reasoning behind how the model arrives at its decision is not explained and therefore users sometimes may find it difficult to understand potential issues with such systems (Dwivedi et al., 2023)

4. **Influence/bias**

   Responses and outputs from generative AI could be inclining towards a particular direction and topic. It could also favour an individual or a group (Ntoutsi et al., 2020). This is when the data used to train models and data in general represents a certain area of issues and context only instead of the overall consensus. Therefore, the data used in training these code generation models can be biased or influenced (Zhuo et al., 2023).

So, in this particular research the author focusses on the aspect of the AI-generated code produced by such generative AI and the bias leading to code quality issues.

2.3.3 AI-generated code

According to the 2023 report from Sonatype, 97% developers and leads use generative AI tools. GitHub also mentioned that developers are not well educated on insecure code and ChatGPT generated a significant amount of vulnerabilities (Hamer, d'Amorim and Williams, 2024). Another recent survey from GitHub, in collaboration with Wakefield research found that 92% of developers from 500 non-student employees who work in companies with over a workforce of 1000+ people use code generation tools for both personal use and work (Staff, 2023).

This leads to believe that the future of developers having majority of their code generated and existing systems and tools are not adapting this change. There are no existing studies on ChatGPT-generated code in context to code quality issues derived from prompts by real developers (Siddiq et al., 2024).

Eventhough code generation tools can be very helpful with problem solving and code generation based on prompts, the output accuracy or results with quality can not be guaranteed. This is because the may use old built-in functions which are vulnerable and old code with exploits (Taeb, Chi and Bernadin, 2024). This means existing systems need to update their data and algorithms to cater to this evolving field.

2.3.4 Code Quality

Code generation tools are being widely developed and used in the industry. However, the output is not very reliable and contains multiple issues in context to code smells and severe issues according to the level of code quality (Yetiştiren et al., 2023). This includes common tools such as ChatGPT, GitHub Copilot and Code Whisperer from Amazon. AI-generated code especially ChatGPT commonly contains quality issues such as incorrect outputs, underperforming, maintainability and compilation errors (Y Liu et al., 2023).

Current ways in identifying code quality issues include extracting data from Json files which contain ChatGPT-generated code and run it against static analyzer tools such as CodeQL, Pylint and Bandit. This way, highlighted issues are recorded which is not very efficient (Siddiq et al., 2024).

There is no work regarding research based on code quality issues in generative AI tools. ChatGPT maybe able to tackle some of the challenges it faces using prompting to improve code quality however there are so many challenges and limitations (Liu et al., 2023).

2.5 Existing work

Static Analysis Tools

When code is written by a developer, a static code analyzer (SCA) will check against predefined custom rules and defined coding rules from the tool. It will then identify whether the code abides with the set of rules. Static analysis tools in the industry these days are essentially bug finding tools that's widely available, incorporated into practically all major integrated development environments (IDEs), and capable of locating a wide variety of defects and issues with code quality. Even while these bug finding tools are quite good at monitoring the quality of the code, most of them rely on custom rules that are constructed on top of basic analyses, such patterns on the abstract syntax tree or data flow inside a single function in the code, rather than being accurate or sound analyzers (Raychev, 2021).

Some recent trends involve using scripts to interact and obtain results from ChatGPT which returns a Json file with the output. This contains information regarding the prompts and code generated. Then the extracted code from json files is run against static analysis tools such as Pylint, Bandit and CodeQL. This process is considered time consuming and are not efficient in obtaining code quality information (Siddiq et al., 2024)

Reflection on Static Analysis Tools

Current tools are built on top of pre-defined templates and rules which check for specific code quality issues and standards. With the current increase in the number of users in terms with generative Ai which was discussed earlier, the tools need to adapt and counter new issues faced.

Some of the main challenges that the static analysis tools according to (Vassallo et al., 2020) include:

1. **Comprehensibility of Alerts**
   The developers sometimes find it difficult to understand the alert message addressed from reported issues as they are sometimes misleading and not easy to understand.

2. **Configuration**
   It is time-consuming to setup such tools especially when it comes to configuring custom rules and templates. This is further supported by deciding which rules can be applicable or inapplicable to the scope of the project.

3. **Effectiveness**
   Some static analysis tools are said to be incompetent in finding the bugs in some types of the problem and that is why the developers overlook them at first. Moreover, vague generalized rules are used which makes it difficult to generalize and it is prone to subjectivity.

Machine learning based approaches

ML based SCA for Software QA (Sultanow et al., 2018): This paper improved to aim the current approach of static analysis tools which typically use basic analyzing such as identifying patters in ASTs (Abstract Syntax Trees) or the flow patterns within a function or method. This paper proposed incorporating ML algorithms and techniques which identify patterns, deviations and standardization from irregularities to improve their accuracy because current SCA tools commonly output false positives which is a significant challenge.

The author makes use ARFF files to generate models based on the patterns identified in the code updating the rule set and the model constantly through reinforcement learning. The paper highlights the fact that the author has tested the performance of some machine learning classifiers such as "MultiLayerPerceptron", "Ibk" and "ADABoost" on a "BFTree" Furthermore, the author of this study also explains the role of feature selection, which acts like a process of selecting more relevant features from the source code in order to help classify an instance as either faulty or bogus. The author also speaks about the parser as a prerequisite fin order to constantly update the model which is capable of detecting faults. Therefore, this paper proposes a feasible solution to improve SCA tools, however there is no proper testing criteria or thresholds which can be used for benchmarking.

Detecting Code Smells using ML (Di Nucci et al., 2018): This paper aims to bridge the subjectivity gap of tools where existing tools are prone to bias and influence. The authors look into and critically analyze six ML techniques such as Random Forest and J48, Naive bayes, Support Vector Machine, B Splines, Bayesian networks and Decision Trees where 4 of them were taken to be evaluated to produce a solution where the grid search algorithm was applied for e4ach ML technique. Then the dataset preparation and preprocessing was done to identify 4 types of code smells and was trained.

The findings however proved to be 90% less accurate that previous studies in terms of F1-score this was mainly due to the reality and nature of the configured dataset may not have reflected real world scenarios. Therefore, threatening the validity and generalizability of the results obtained.

Deep learning based approaches

In today's world, Machine Learning (ML) is an important element which shows how artificial intelligence is doing similar to what humans think and learn. Deep Learning (DL) is one of the major components of Machine Learning which mimics the multi layered architecture and the operations conducted by the human brain. DL techniques are more and more widespread recently due to rise of big data and having performance better than traditional ML techniques is the factor which makes it a really famous method nowadays (Alzubaidi et al., 2021).

Detecting Code Smells using DL (Das, Yadav and Dhal, 2019)**:** The significance of this research is that it is a new approach to detecting code smells using DL which achieved a high accuracy. It includes multiple layers such as convolution, flattening and dense layers that use feature metrics which are then fed into a CNN model for training and classification.

However, there could be other metrics which can be used which may provide more accurate results. The authors don’t mention how the metrics that they chose were the best approach. This study is limited to detecting only two code smells (Brain Class and Brain Method) and it addresses only Java projects which may not be generalizable to other datasets and findings for other programming languages. It also relies on a specific set of metrics to represent the code whereas other metrics may be more effective in detecting code smells. Therefore, these could be considered as some of its limitations.

Fusion of deep convolutional and LSTM recurrent neural networks for automated detection of code smells (Ho et al., 2023)**:** Combines deep CNNs and long short-term memory networks (LSTM) to capture both structural and semantic features of code fragments for automatically detecting code smells which is an innovative approach. The final classification is conducted by deep neural networks with a weighted loss function to reduce the impact of skewed data distribution. The authors also have conducted an empirical study against benchmark datasets which validate their findings.

Four types of code smells were targeted where 4 different datasets were used in order to tackle the 4 different types of code smells. Using different datasets for predicting the different code smells could introduce bias. Moreover, the findings of this study maybe valid only for the considered datasets as it may not be generalizable for a real-world solution as it targets only specific smells.

2.6 Technological Review

2.6.1 Metric-based code quality issue detection

Multiple approaches to detect issues in code have been proposed from early times so lets start with one of the most initial methods which is still being used today. This includes traditional ways by using heuristics which is by creating templates that include rules that the expected code should follow and will be compared and run against (Pecorelli et al., 2019).

It works by following two main steps: (i) Includes identifying the root cause or symptom which is mapped to certain thresholds based on metrics determined by structure. (LOC >= k, i.e. when lines of code are greater than or equal to k). (ii) Combining the root cause with the structure to identify the most suitable rule template which the code then runs against (Moha et al., 2010) which have been followed throughout a long period of time that lead to 3 major issues such as subjectiveness, no proper benchmarking between these tools and difficulty to identify common thresholds to compare results (Pantiuchina, Lanza and Bavota, 2018).

2.6.2 Machine learning based code quality issue detection

Majority of the current detection systems use metric based systems on structural code and existing machine learning techniques are not configured correctly meaning there is no standard way to preprocess data, select features and no availability of a standard dataset (Azeem et al., 2019).

Moreover, there is no to little work in applying deep learning approaches to detect code quality issues. Recent deep learning approaches to identify brain method and class code smells have been promising to be accurate (Das, Yadav and Dhal, 2019).

In order to eliminate code quality issues, it roughly took around 540 seconds using Copilot and 340 seconds using Amazon CodeWhisperer and around 520 seconds using ChatGPT through prompting (Yetiştiren et al., 2023). So the author aims to tackle these timings with a novel approach.

2.6.3 Data preprocessing

Refers to the initial stage of data analysis where the data is explored, cleaned, filtered and only the essential features are a kept for further processing. This helps solve many challenges such as null/missing values, outliers, scaling, transformation and data imbalance (Fan et al., 2021). This would help researchers to explore their data in depth and obtain insights prior to training a model

2.6.4 Tokenization

Tokenization involves breaking down larger volumes of text into smaller parts or essentially its subwords or 'tokens'. This is usually done by separating any prefixes and suffixes prior to the training process. This is an important steps to allow machines to learn without getting overwhelmed by large volumes of data and understand meaning (Alkaoud and Syed, 2020)

2.6.5 Feature Selection

Feature selection involves identifying the most appropriate and relevant inputs(features) for the model removing any unnecessary and redundant features. Getting stable and high accurate results directly corrlates to the feature selection algorithm followed so it is a crucial step for the researches in this domain (Khaire and Dhanalakshmi, 2022).

2.6.6 Hyperparameter tuning

Hyperparameter tuning has not been used in prior techniques and implementation for previous works. So, the author would explore various possibilities by themselves.

Most parameters in machine learning algorithms are configured by default. The manual change of these parameters for optimal performance is known as the tuning procedure which may involve research, trial and error. Applying and assessing the right hyperparameters enable to achieve significant optimal results depending on the context of studies (Probst, Boulesteix and Bischl, 2019)

2.6.7 Dataset challenges

The author is the first to explore code quality issues in AI-generated code, therefore there is a scarcity of data. Only the dataset published by (Liu et al., 2023) including around 2000 programming tasks from leetcode which contained information about their conversation with ChatGPT and issues in them seemed ideal. However, it has to go through a strenuous data preprocessing to extract data from Json files and convert it into a format that can be used for model training.

2.7 Evaluation and benchmarking

2.7.1 Evaluation Metrics

Evaluation Metric	**Description**
Accuracy	Accuracy measures how correctly the model classifies the code quality issues from the code quality issue sample. Higher values indicate better performance for the model
F1 Score	From 0 being the worst to 1 being the best, it is a balanced measure which takes the average of recall and precision. It is a

	more acceptable measure that just taking precision and recall individually as it proves the robustness and reliability of the model
Precision	Used to evaluate the instances where the model is predicting the positive labels. In the context of this project, how correct were the predictions from all the positively predicted values.
Recall	Recall is used to evaluate the positively predicted labels. This is used to identify how much of the identified true positives are actually true.

Table 2: Evaluation metrics

Previous work suggest that using precision, recall and F1-score is a ideal way to evaluate deep learning models in this domain as it helps identify the overall performance of a mode from all viewpoints (Ho et al., 2023).

2.7.2 Benchmarking challenges

Since the author is the first to explore code quality issues in AI-generated code, there is no standard benchmarking dataset to compare findings. Existing benchmarks in similar domains are outdated in context to AI code generation as they were released before 2021(Chen et al., 2021). These lacked subsequent and appropriate metadata for taks so they are not ideal for the domain of this research because modern tools with the rapid increase of generative AI after the release of ChatGPT provide a more realistic user relatability (Liu et al., 2023)

2.8 Chapter summary

This chapter dived deeply into the overall literature of the research area. The author explored the problem domain in depth followed by the current mechanism used in the industry. Afterwards, exiting work was critically evaluated and inspected. Towards the end of the chapter, technologies were reviewed along with evaluations and benchmarking for this domain were covered. The next chapter will traverse through the methodologies for the project

CHAPTER 03: METHODOLOGY

3.1 Chapter Overview

This chapter will cover the research, design, development, evaluation, solution and project management methodologies chosen by the author for this research project along with its available options and justifications as to why the author chose particular methodologies. Additionally, the project scopes will be discussed along with the planned schedule and deliverables. Later in the chapter, the hardware, software, data and skill requirements required for the project will be explored along with the risks and mitigations to be anticipated.

3.2Research Methodology

Research Philosophy	Out of the 4 research philosophies: positivism, interpretivism, pragmatism and realism, **Pragmatism** was chosen. This is because the author will evaluate both quantitative and qualitative methodologies to experiment and determine which would produce best performance in achieving the research goal.
Research Approach	As this research aims to comprehensively apply a variety of theories and evaluate the hypothetical statement – the **deductive** approach was chosen out of the two available research approaches which are inductive and deductive.
Research Choice	From the given options: mono method, multi method and mixed method – the **mixed method** was picked. This is because both methods – qualitative and quantitative are used through the intended use of interviews, experiments and surveys to collect data throughout the research process and evaluate.
Research Strategy	**Surveys**, **experiments** and **interviews** will be conducted amongst potential candidates based on performance and evaluation metrics. The

	research strategy would help address and provide insights into answering the research questions of the project.
Time Horizon	Amongst the available time horizons namely: longitudinal and cross-sectional, the **cross-sectional** approach was chosen because evaluations and data collection would be carried out on one point in time during the evaluation phase of the project.
Data Collection and Analysis	Interviews, surveys, trial and error observations along with multiple other techniques would be used in order to collect and analyze data. Also, insights gained from similar work and literature will be applied.

Table 3: Research methodologies

3.3Development Methodology

The **prototype** development methodology was chosen amongst the various software lifecycle development methodologies because this project aims to develop, design and evaluate the prototype as needed and accordingly to new findings and evaluations in the industry as it's an emerging and rapidly growing research area.

3.3.1 Design Methodology

Object Oriented Analysis and Design (OOAD), out of the available design methodologies, was chosen. In order to adapt to frequent changes and updates in the industry, especially with LLMs and AI-generated code, new information will be gathered and updated regularly throughout this research project. Therefore, in order to accommodate the need for constant changes and requirements, this method was chosen for its adaptable and iterative aspects.

3.3.2 Evaluation Methodology

The following evaluation metrics were chosen by the author from the deep learning and code analysis domain in order to quantitively evaluate the performance:

- Precision and Recall
- F1 Score
- Classification Accuracy

- ROC curve and AOC

3.3.3 Solution Methodology

The dataset that contains only AI-generated code needs to be gathered during the primary and secondary data collection process in order to identify code quality issues specific to AI-generated code. Then the data should move into the data processing stage where the data cleaning and handling of missing data is done. After the feature engineering and model training, the algorithm is chosen after fine tuning through evaluation, the model is then deployed.

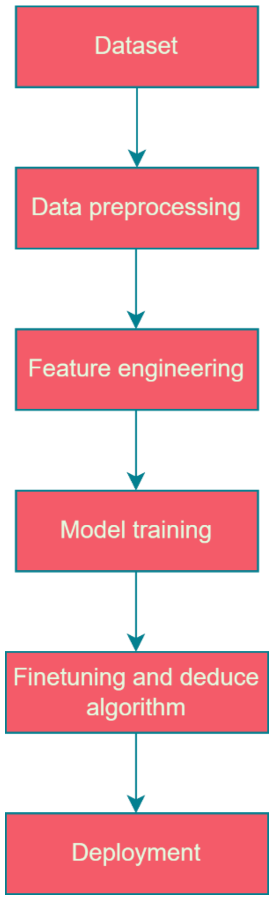


Figure 2: Workflow diagram

3.4 Project Management Methodology

From the available options **Agile Prince 2** methodology was chosen for project management. Agile Prince 2 allows the author to be more open and flexible to improvements and swift changes which is a crucial aspect to the execution of this research project.

3.4.1 Project Scope

3.4.1.1 In-scope

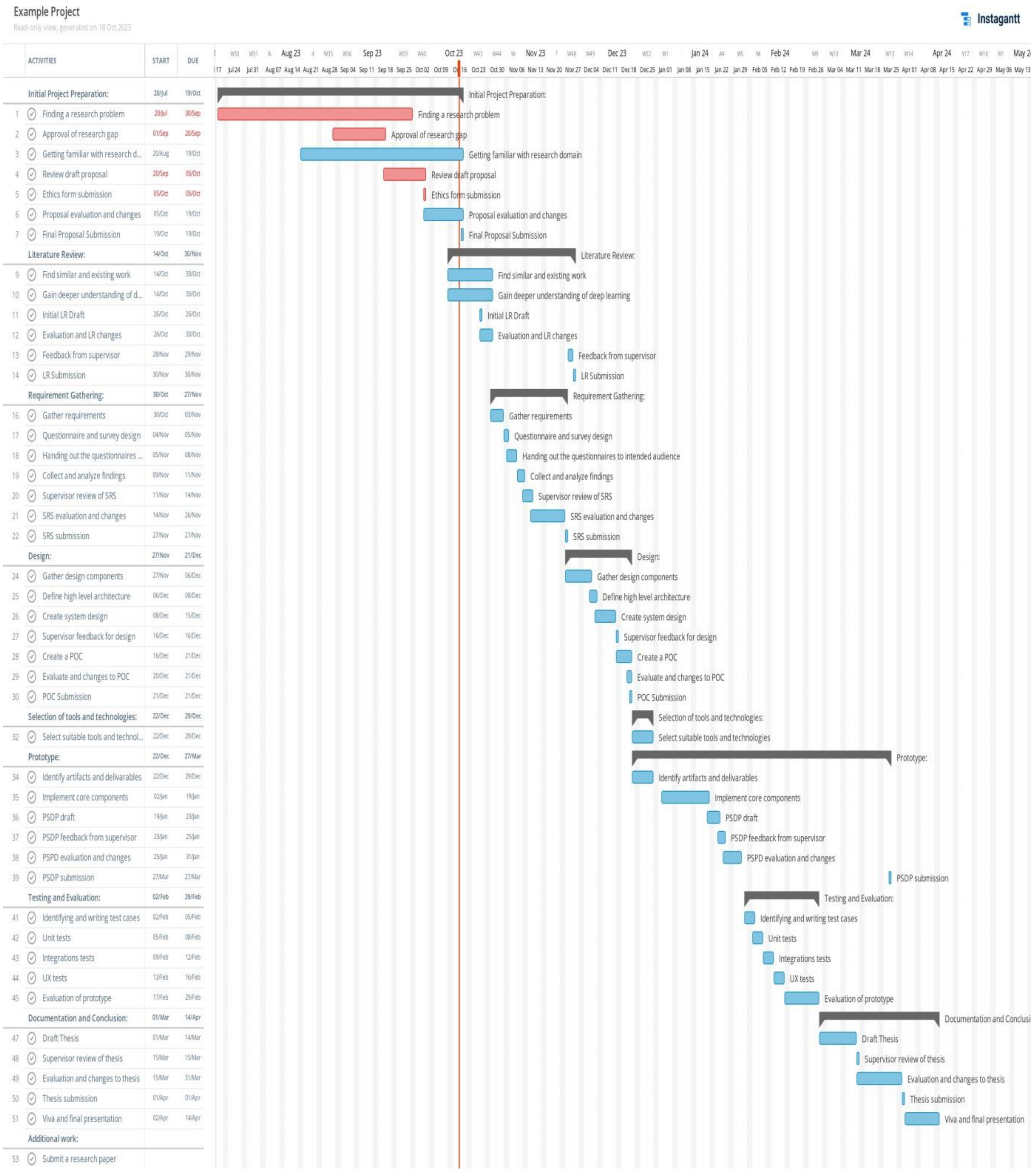
- Developing an algorithm to detect code quality issues in AI-generated code.
- A method to classify different types of code quality issues.
- A web interface to display and evaluate the proposed deep learning solution.

3.4.1.2 Out-scope

- The proposed solution will only detect code quality issues.
- Built for only limited programming languages.
- Produce a prompt that would automatically resolve the issues in the code.

3.4.2 Schedule

3.4.2.1 Gantt Chart



Example Project

Read-only view, generated on 18 Oct 2023

	ACTIVITIES	START	DUE
	Initial Project Preparation:	20/Jul	19/Oct
1	Finding a research problem	20/Jul	30/Sep
2	Approval of research gap	01/Sep	20/Sep
3	Getting familiar with research d...	20/Aug	19/Oct
4	Review draft proposal	20/Sep	05/Oct
5	Ethics form submission	05/Oct	05/Oct
6	Proposal evaluation and changes	05/Oct	19/Oct
7	Final Proposal Submission	19/Oct	19/Oct
	Literature Review:	14/Oct	30/Nov
9	Find similar and existing work	14/Oct	30/Oct
10	Gain deeper understanding of d...	14/Oct	30/Oct
11	Initial LR Draft	26/Oct	26/Oct
12	Evaluation and LR changes	26/Oct	30/Oct
13	Feedback from supervisor	28/Nov	29/Nov
14	LR Submission	30/Nov	30/Nov
	Requirement Gathering:	30/Oct	27/Nov
16	Gather requirements	30/Oct	03/Nov
17	Questionnaire and survey design	04/Nov	05/Nov
18	Handing out the questionnaires ...	05/Nov	08/Nov
19	Collect and analyze findings	09/Nov	11/Nov
20	Supervisor review of SRS	11/Nov	14/Nov
21	SRS evaluation and changes	14/Nov	26/Nov
22	SRS submission	27/Nov	27/Nov
	Design:	27/Nov	21/Dec
24	Gather design components	27/Nov	06/Dec
25	Define high level architecture	06/Dec	08/Dec
26	Create system design	08/Dec	15/Dec
27	Supervisor feedback for design	16/Dec	16/Dec
28	Create a POC	16/Dec	21/Dec
29	Evaluate and changes to POC	20/Dec	21/Dec
30	POC Submission	21/Dec	21/Dec
	Selection of tools and technologies:	22/Dec	29/Dec
32	Select suitable tools and technol...	22/Dec	29/Dec
	Prototype:	22/Dec	27/Mar
34	Identify artifacts and delivarables	22/Dec	29/Dec
35	Implement core components	02/Jan	19/Jan
36	PSDP draft	19/Jan	23/Jan
37	PSDP feedback from supervisor	23/Jan	25/Jan
38	PSPD evaluation and changes	25/Jan	31/Jan
39	PSDP submission	27/Mar	27/Mar
	Testing and Evaluation:	02/Feb	29/Feb
41	Identifying and writing test cases	02/Feb	05/Feb
42	Unit tests	05/Feb	08/Feb
43	Integrations tests	09/Feb	12/Feb
44	UX tests	13/Feb	16/Feb
45	Evaluation of prototype	17/Feb	29/Feb
	Documentation and Conclusion:	01/Mar	14/Apr
47	Draft Thesis	01/Mar	14/Mar
48	Supervisor review of thesis	15/Mar	15/Mar
49	Evaluation and changes to thesis	15/Mar	31/Mar
50	Thesis submission	01/Apr	01/Apr
51	Viva and final presentation	02/Apr	14/Apr
	Additional work:		
53	Submit a research paper		

Instagantt
Aug 23
Sep 23
Oct 23
Nov 23
Dec 23
Jan 24
Feb 24
Mar 24
Apr 24
Getting familiar with research domain
Gain deeper understanding of deep learning
Handing out the questionnaires to intended audience
Select suitable tools and technologies
Documentation and Conclusi

Figure 3: Gantt Chart

3.4.2.2 Deliverables

Deliverable	Date
Final Project Proposal	19th October 2023
Literature Review Thorough analysis and critical evaluation of existing and related work in the domain.	30th October 2023
System Requirement Specification Specifies the requirements to be met by the proposed prototype.	27th November 2023
Proof of Concept Submission An initial draft implementation of the proposed solution.	21st December 2023
Final Project Specifications Design and Prototype (PSDP) Documentation and prototype of the implemented solution along with its core features.	1st February 2024
Minimum Viable Product Submission	7th March 2024
Final Thesis Submission	1st April 2024

Table 3: Deliverables

3.4.3 Resource Requirements

In order to meet the expectations and requirements of the aforementioned methodologies and data gathering techniques, the skills, software, hardware and data requirements are as follows:

3.4.3.1 Hardware Requirements

- **Intel I5 or I7 CPU** – To handle high processing power to train and test deep learning models.
- **8GB RAM or above** – To control and manage huge datasets

- **50GB free disk space** – To store datasets and models prior and post training along with application data.

If hardware requirements are not met or sufficient, the recommended approach of using cloud technologies such as google collab would be used.

3.4.3.2 Software Requirements

- **Python** – Efficient to use for machine learning and deep learning projects as it allows effective training of models.
- **React** – To build the front end of the proposed prototype and provide good user experience.
- **Zotero** – To organize, build and manage research papers and citations.
- **GitHub** – Sufficient and organized version control.
- **Google Drive** – To keep regular backups of the project.
- **MS Word/Google Docs** – To provide clear documentation.
- **Jupiter Notebook/PyCharm IDE** – To develop and maintain code sufficiently.
- **Figma** – To create and design UI prototypes to display the prototype.

3.4.3.3 Data Requirements

In order to build a successful model, datasets that contain AI-generated code would be required which is readily available from Codex and figshare. This can be used to test and evaluate the proposed prototype.

3.4.3.4 Skill Requirements

- Machine Learning/Deep Learning expertise – A thorough understanding of ML and DL techniques and theory are required. Domain knowledge of code analysis and its link to ML and DL is also a vital aspect for this research project.
- Ability to design user friendly interfaces to display the prototype.
- Presentation skills to pitch and present the project adequately.
- Documentation skills to create and maintain proper standard documentation for the prototype and provide the project deliverables.

3.4.4 Risks and Mitigation

Risk Item	Severity	Frequency	Mitigation Plan
No sufficient knowledge on technical and domain areas.	5	5	Read and gain insights on domain area from experts.
Losing documentation, data and code.	4	2	Take regular backups and maintain a GitHub repository.
Lack of hardware resources	4	3	Switch to cloud-based solutions and development environments such as Google Colab.
Project requirements changing due to the project being a part of a rapidly growing domain.	4	5	Stay updated on the latest changes by staying in touch with domain experts and latest research papers.
Hypothesis turning out to be contradicting or invalid.	4	3	Continue and keep exploring the research field as any strong output would be a contribution.

Table 4: Risks and mitigation

3.5 Chapter Summary

In this chapter, the methodologies followed by the author for this project were discussed. Diving into the research methodologies and the prototype development along with the OOAD design approaches were chosen amongst the available options. The evaluation metrics used for assessing the project were outlined. Moreover, the solution methodology used for a DL approach was introduced with the integration of Agile Prince-2 project management with the justifications as to why they were chosen. Additionally, this chapter explored the project scopes covered, schedules, deliverables, required skills, data and resource needs and the risks and mitigations that should be anticipate.

CHAPTER 04: SOFTWARE REQUIREMENTS SPECIFICATION

4.1 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.

4.2 Rich Picture Diagram

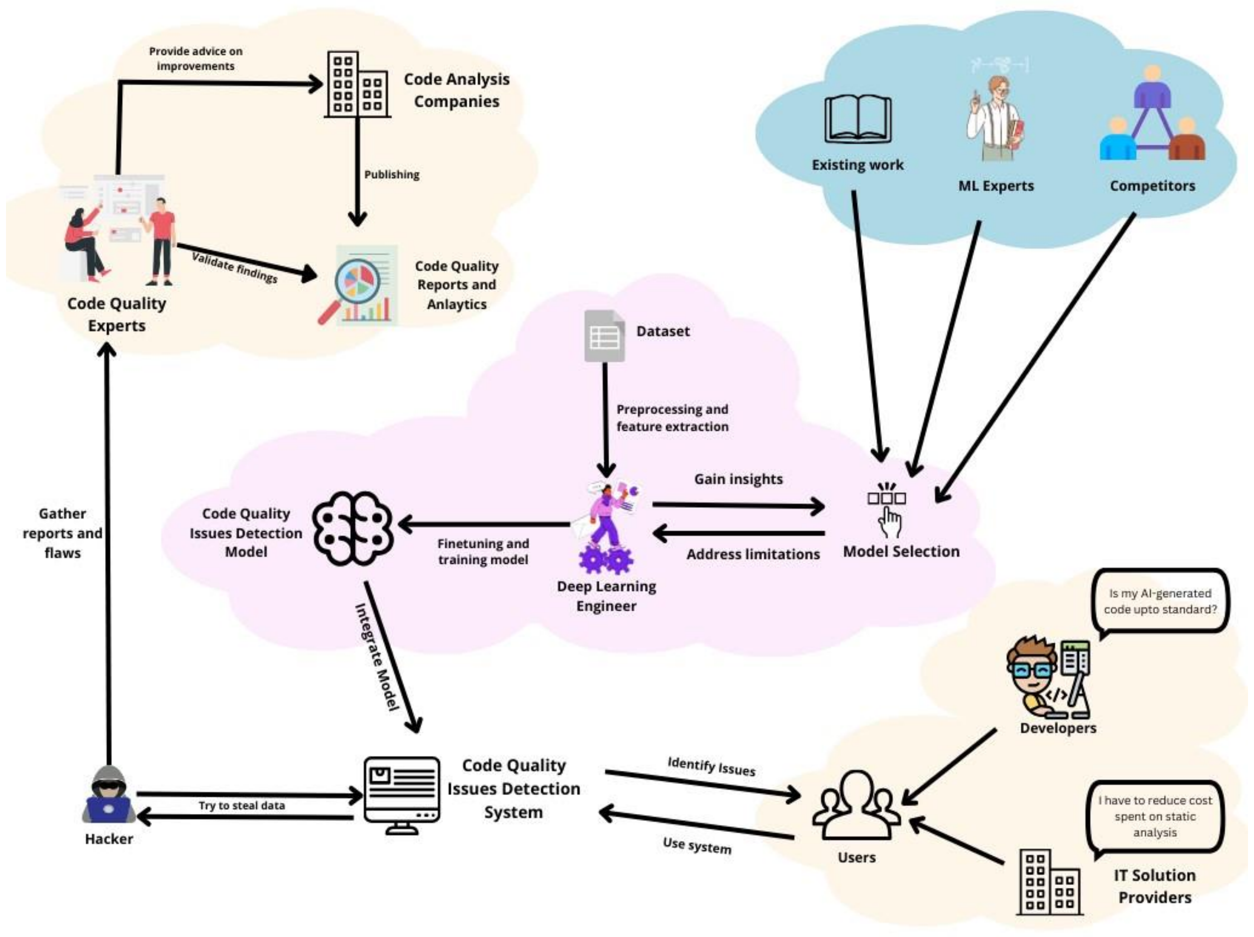


Figure 4: Rich Picture Diagram

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.

4.3 Stakeholder Analysis

4.3.1 Stakeholder onion model

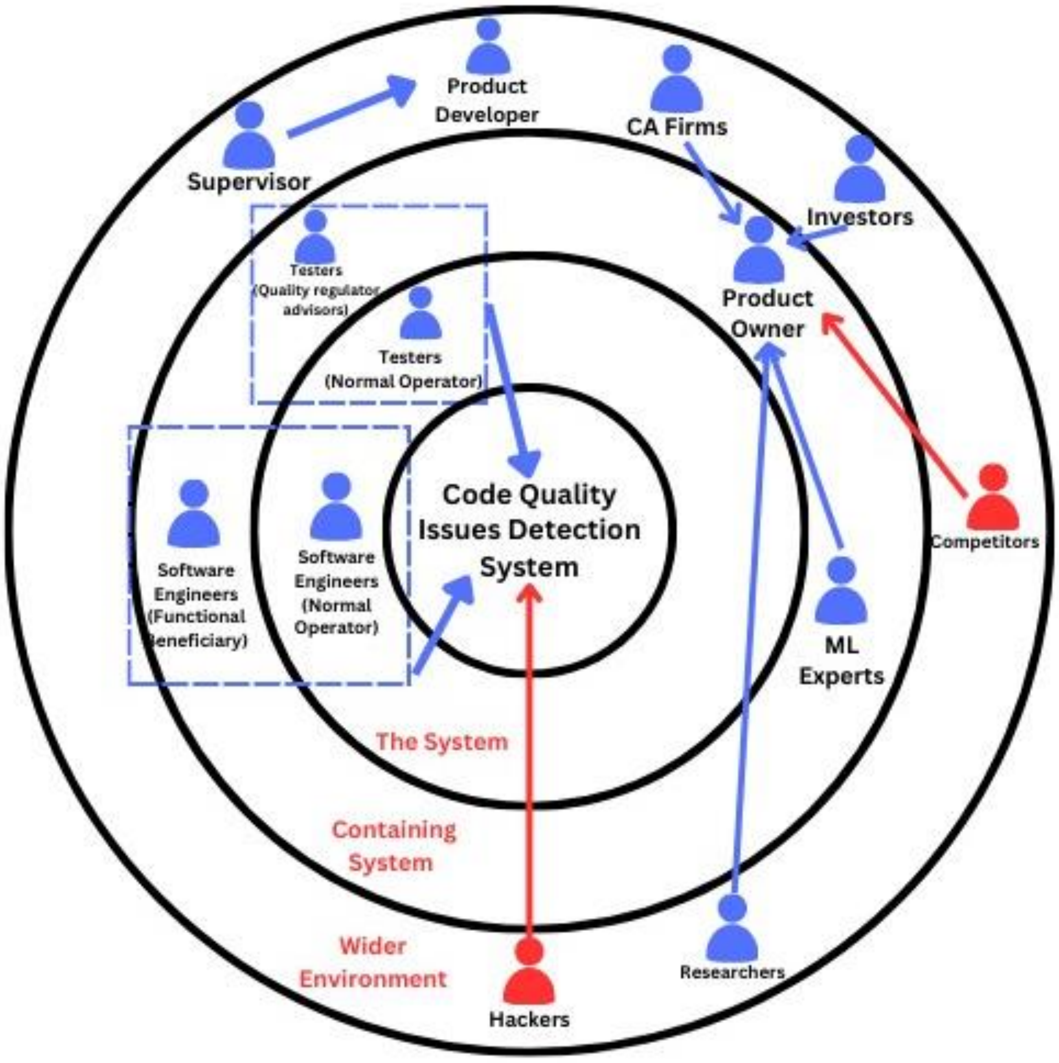

Product Developer
CA Firms
Supervisor
Investors
Testers (Quality regulator advisors)
Testers (Normal Operator)
Product Owner
Software Engineers (Functional Beneficiary)
Software Engineers (Normal Operator)
Code Quality Issues Detection System
Competitors
ML Experts
The System
Containing System
Wider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

Figure 5: Stakeholder Onion Model

4.3.2 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 5: Stakeholder Analysis

4.4 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 6: Requirement Elicitation Methods

4.5 Discussion of Findings

4.5.1 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. Code smells detection such as brain method and brain class identification using CNN models have shown to be accurate
(Kanan, Sherief and Abdelmoez, 2022)	DL approaches produce a higher accuracy in detecting code smells when compared to traditional ML approaches.

Table 7: Literature Review Findings

4.5.2 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 B.

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 DL in detecting code quality issues.

Table 8: *Thematic Analysis for Interviews*

4.5.3 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 DL methods.	✓		
Evaluation metrics such as TP, FP, recall precision can be used to evaluate the model	✓	✓	✓

Table 9: Summary of Findings

4.6 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.

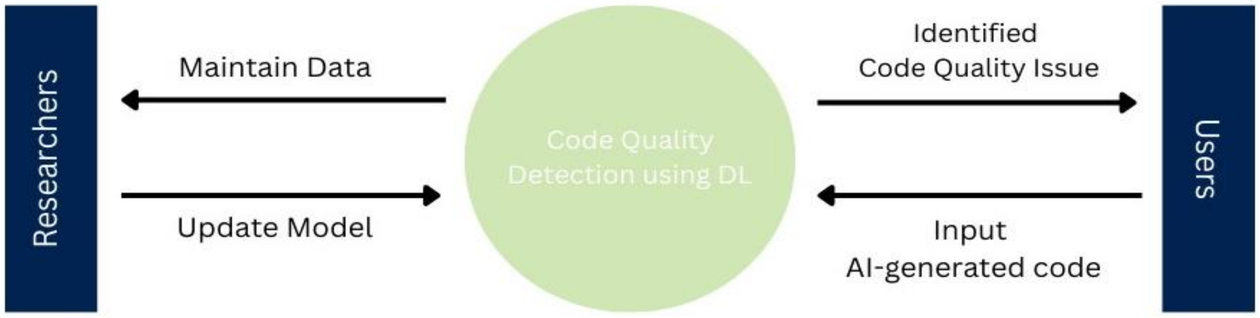


Figure 6: Context Diagram

4.7 Use Case Diagram

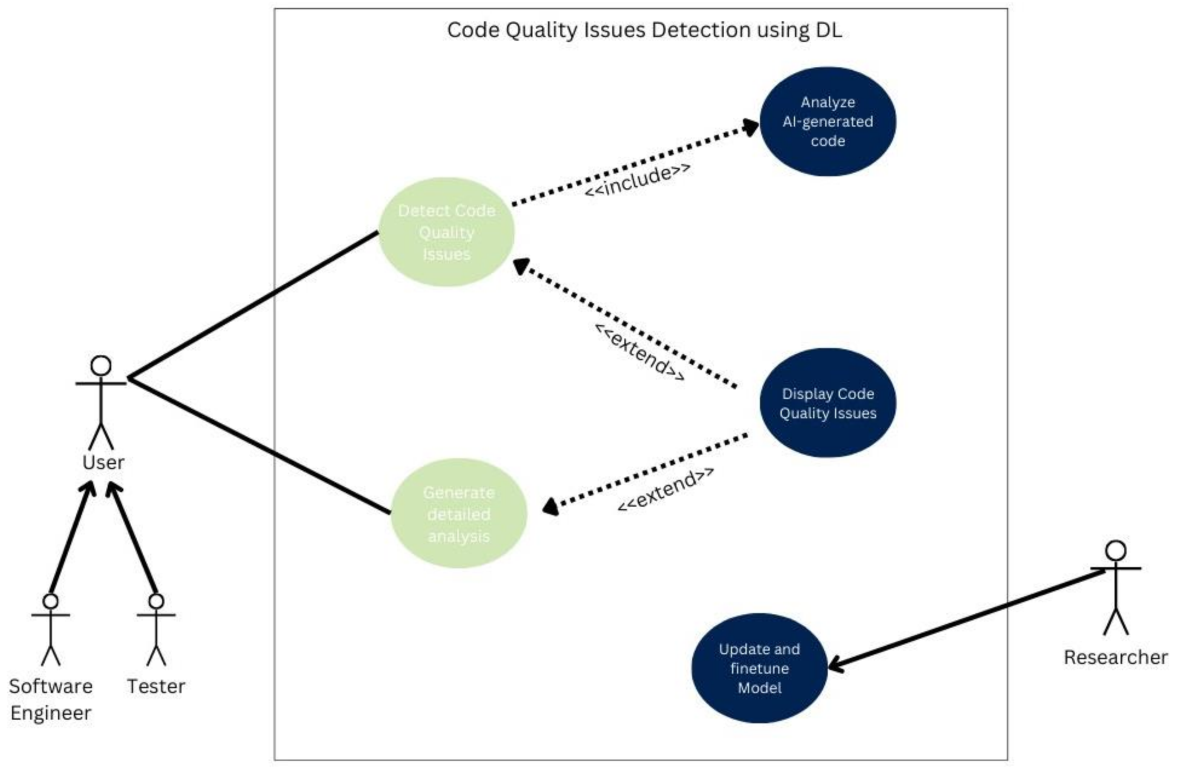

Code Quality Issues Detection using DL
Analyze AI-generated code
<<include>>
Detect Code Quality Issues
<<extend>>
Display Code Quality Issues
<<extend>>
Generate detailed analysis
User
Software Engineer
Tester
Update and finetune Model
Researcher

Figure 7: Use Case Diagram

4.8 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 10: Use Case 1 Description

4.9 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 11: MoSCoW Prioritization Levels

4.9.1 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 12: Functional Requirements

4.9.2 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 13: Non-functional Requirements

4.10 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 05: SOCIAL, LEGAL, ETHICAL AND PROFESSIONAL ISSUES (SLEP)

5.1 Chapter Overview

This chapter explores the SLEP issues that may have raised concerns throughout the course of this project and how the author countered and mitigated these issues.

5.2 Social Issues

- The project is free from any illicit and political agendas. It also does not affect or imply any emotional bias towards the results.
- The project proposal and PSDP documents were presented any evaluators or interviewees who requested more context to the study. A thorough explanation of the ideologies, methodologies and process of the entire research project was explained to them as well.

5.3 Legal issues

- The code entered by the data is not stored and is discarded after the analysis therefore maintaining anonymity and avoiding data leaks. Moreover, no data related to the user's identity or any user related information is gathered.
- The dataset gathered for the project was published with an open-source license and is available for public use.
- The interviewees mentioned in the thesis gave consent prior to the interviews conducted to use their name and designation for this research purpose.

5.4 Ethical Issues

- The thesis does not contain any plagiarized material and all external information was given accreditation through proper citations throughout the thesis.

- The evaluators and interviewees were given a brief overview of their contribution in this study.

5.5 Professional Issues

- Best practices and professional standards were maintained wherever applicable to follow industry standard guidelines. The code proposed follows best practices and guidelines and no illegal tools were used.

- This project does not replace jobs. It is just adapting to the change of generative AI and is a proposed novel approach to update existing systems. The primary focus contributing research in the field of AI-generated code.

5.6 Chapter Summary

The SLEP issues were addressed in this chapter exploring each factor (social, legal, ethical, professional and ethical) in depth and the potential and implemented mitigation approaches were discussed.

CHAPTER 06: DESIGN

6.1 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.

6.2 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 14: Design Goals

6.3 High level Design

6.3.1 Architecture Diagram

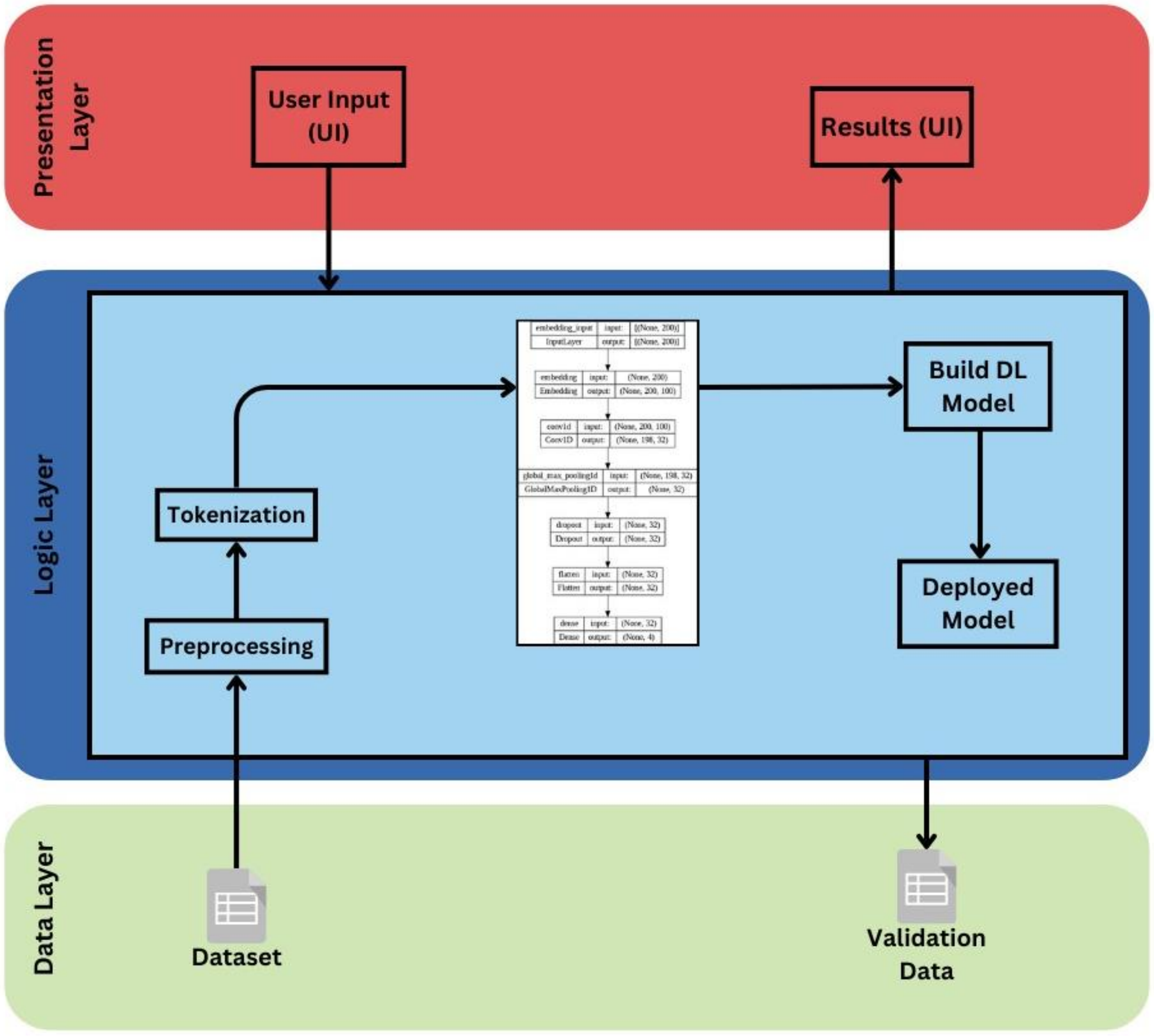

Presentation Layer
User Input (UI)
Results (UI)
Logic Layer
Tokenization
Preprocessing
Build DL Model
Deployed Model
embedding_input | InputLayer | input: [(None, 200)] | output: [(None, 200)]
embedding | Embedding | input: (None, 200) | output: (None, 200, 100)
conv1d | Conv1D | input: (None, 200, 100) | 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)
Data Layer
Dataset
Validation Data

Figure 8: Layered Architecture Diagram

6.3.2 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.

6.4 System Design

6.4.1 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.

6.5 Detailed Design Diagrams

6.5.1 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.

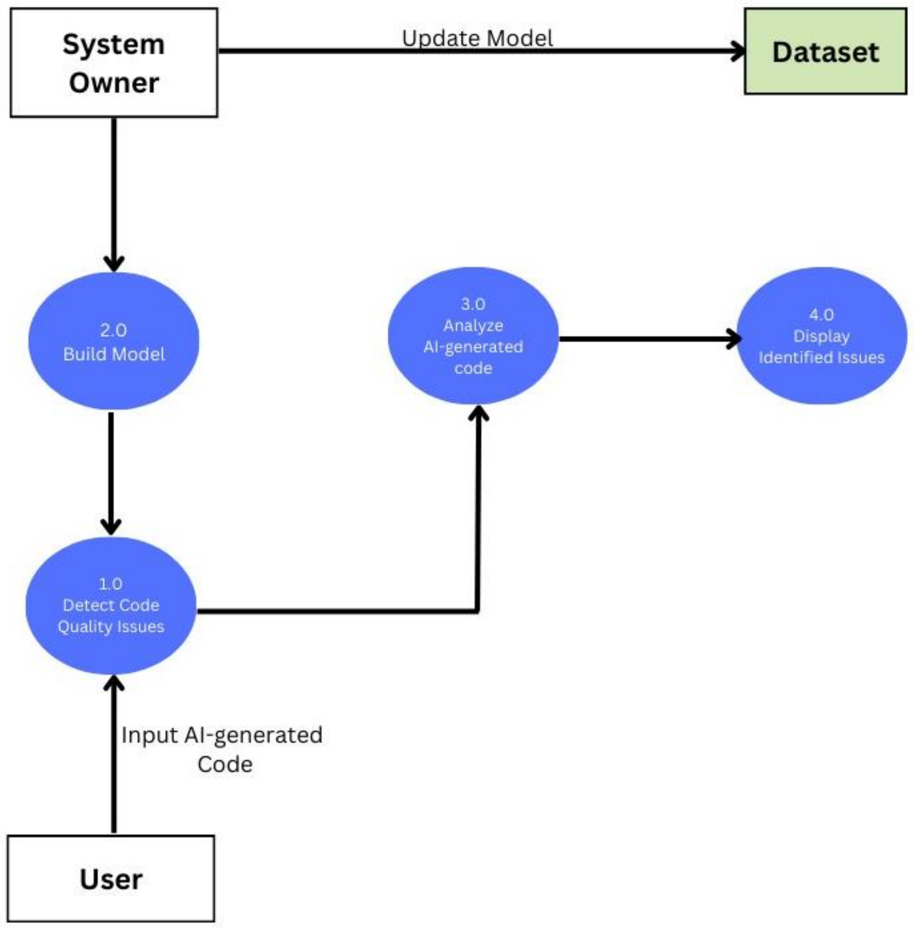

System Owner
Update Model
Dataset
2.0 Build Model
3.0 Analyze AI-generated code
4.0 Display Identified Issues
1.0 Detect Code Quality Issues
Input AI-generated Code
User

Figure 9: Level 1 data flow diagram

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.

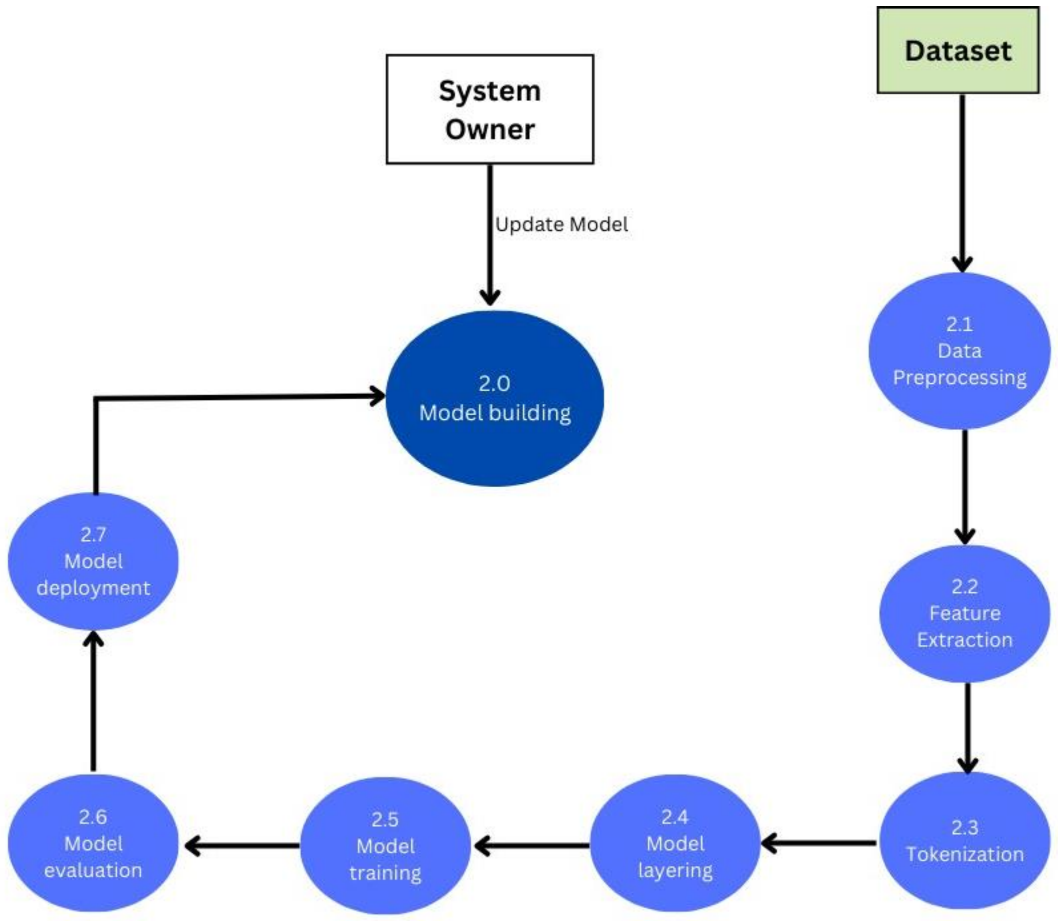
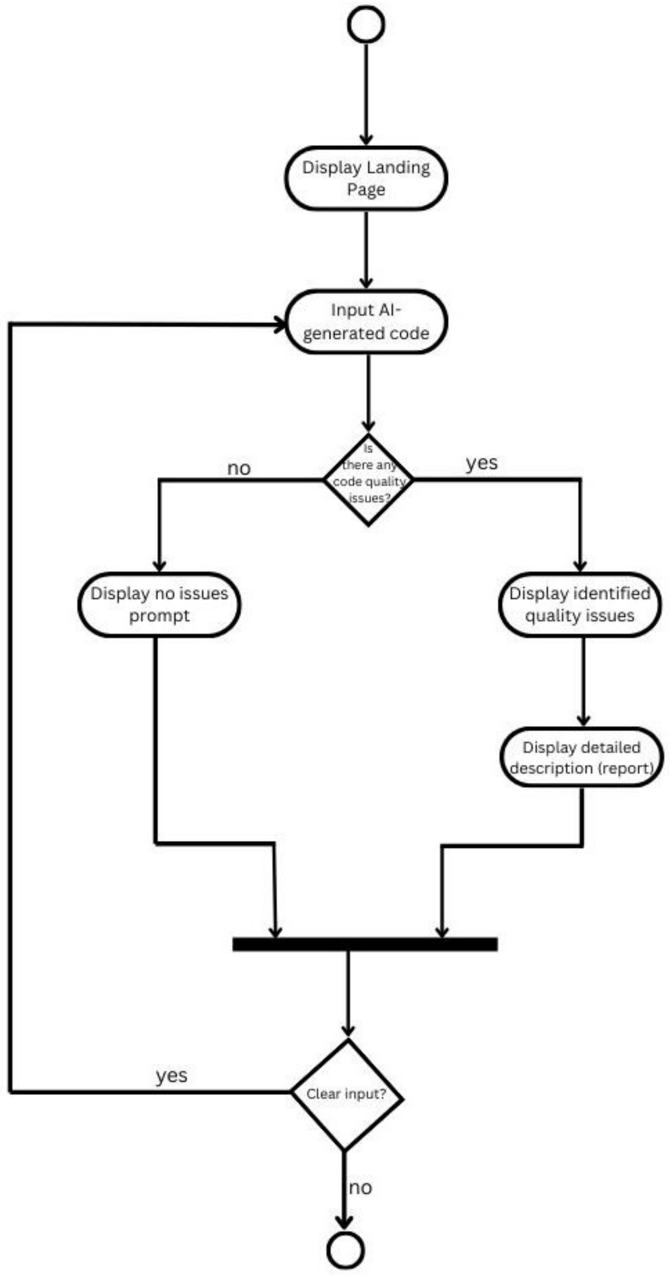


Figure 10: Level 2 data flow diagram

6.5.2 System Process Flow Chart

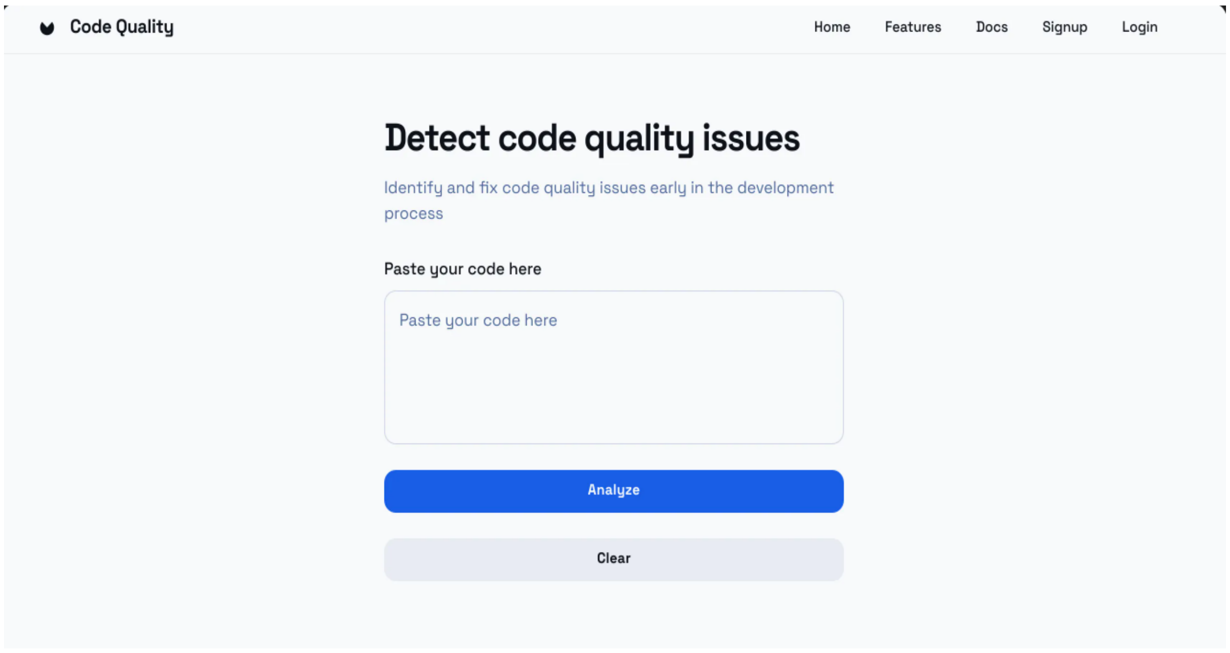
The below illustrated diagram shows the flow of the proposed system. It also shows the order of steps taken to get the desired outcome. This gives a clear visualization to understand the process from start to end in a system.



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

Figure 11: Activity Diagram

6.5.3 User Interface Design



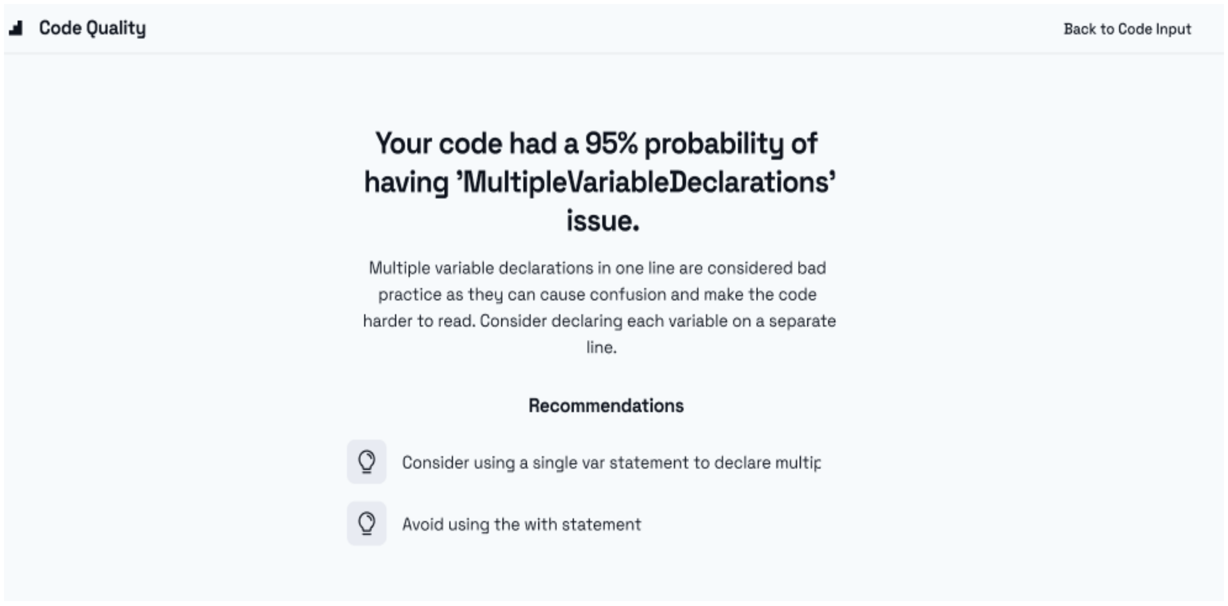
Detect code quality issues

Identify and fix code quality issues early in the development process

Paste your code here

Paste your code here

Figure 12: 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.

Recommendations

Consider using a single var statement to declare multip

Avoid using the with statement

Figure 13: UI for Output Page (Displaying Results)

6.6 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 07: IMPLEMENTATION

7.1 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.

7.2 Technology Selection

7.2.1 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.

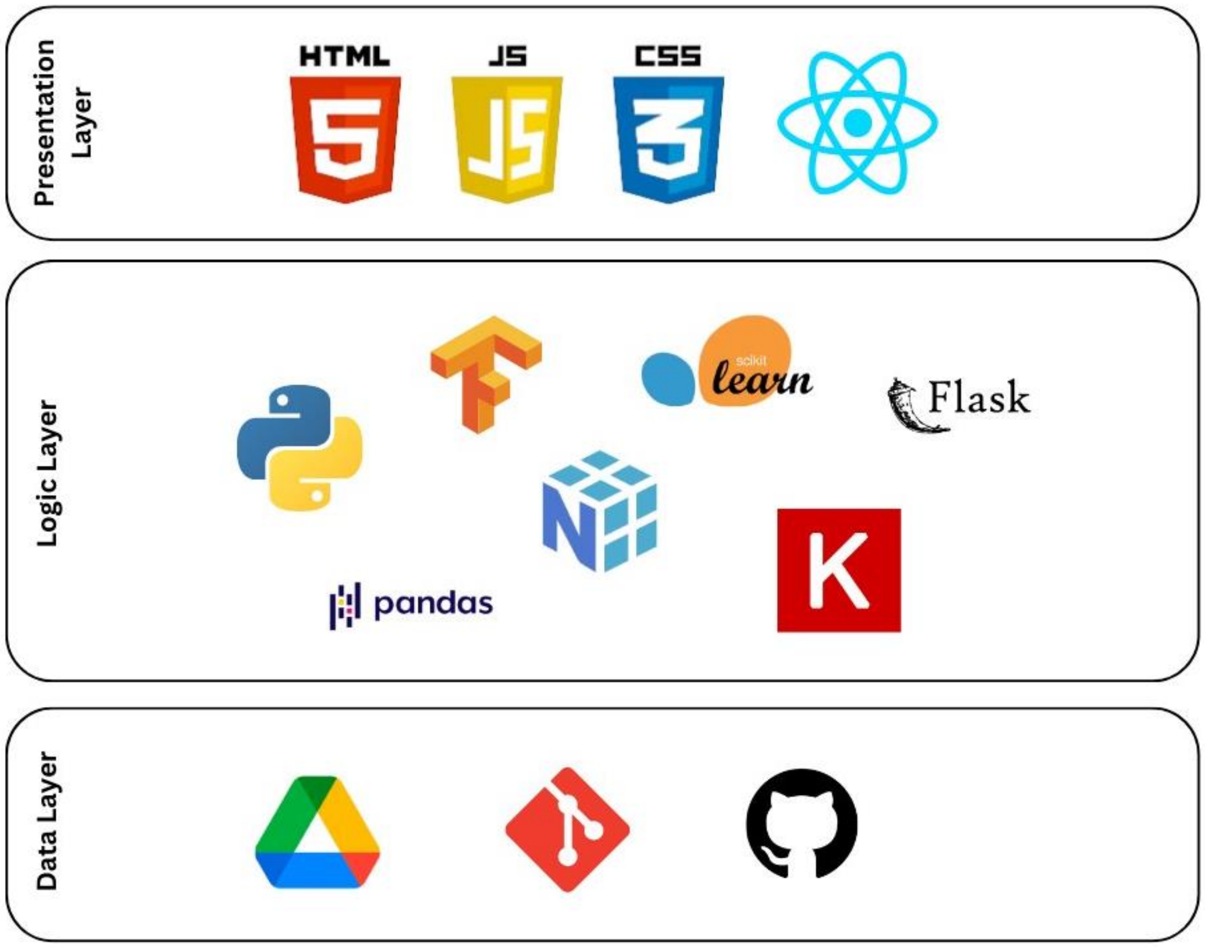


Figure 14: Technologies Used

7.2.2 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).

7.2.3 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 15: Framework Selection Justifications

7.2.4 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 16: Programming Languages Justifications

7.2.5 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 17: Libraries Selection Justifications

7.2.6 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 18: IDE Selection Justifications

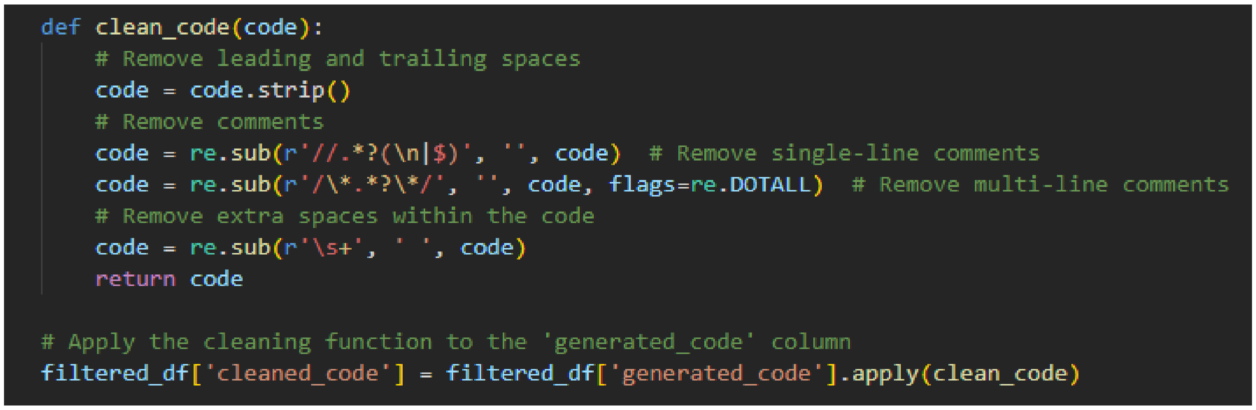
7.2.7 Summary of Technology Selection

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 19: Summary of Technology Selection

7.3 Implementation of the Core Functionality

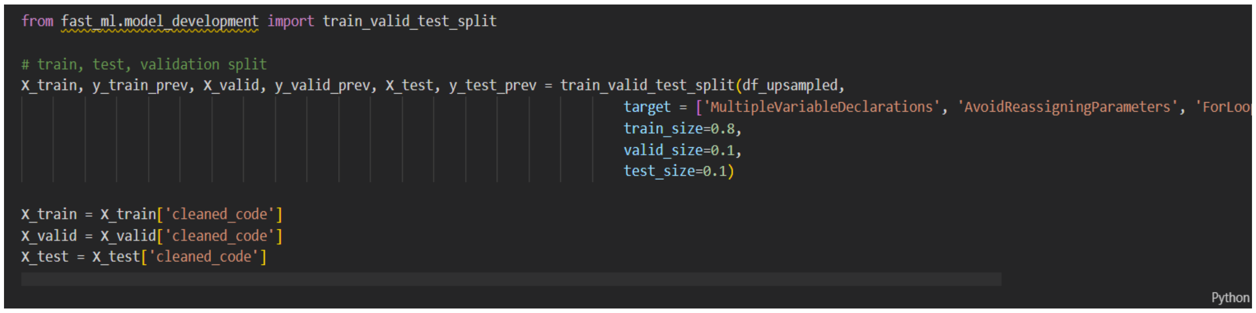
The author initially had to preprocess the data by cleaning the code which involved removing leading and trailing spaces from code, remove comments (single-line and multi-line) and remove unnecessary extra spaces within the code. This is to ensure that the model is not influenced by irrelevant formatting details and focus more on the code itself and its logic. The core and thorough preprocessing steps prior to this core phase to gather required data have been included in GitHub.



```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 15: 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 the data was split into train, test and validation sets respectively and can be visible in the repository link shared below.

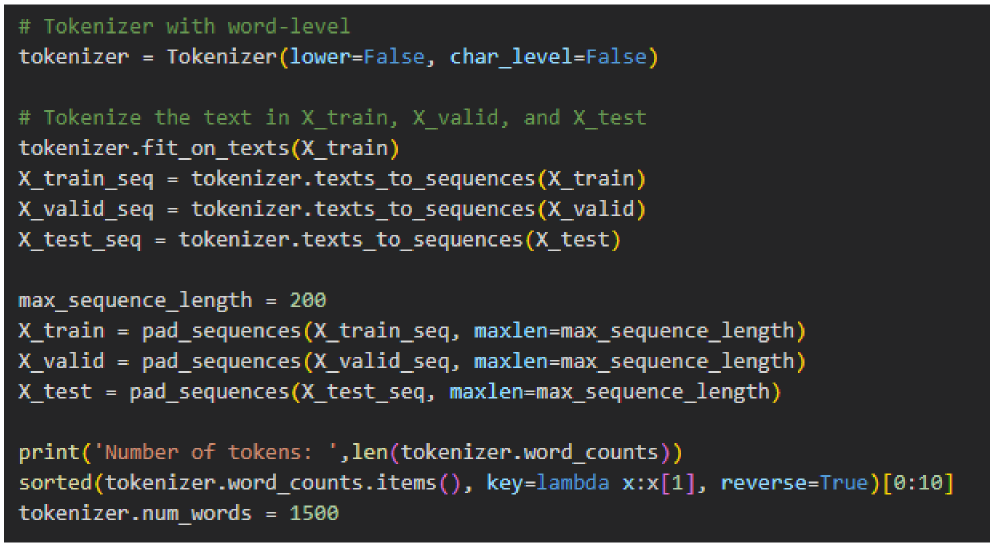


```python
from fast_ml.model_development import train_valid_test_split

# train, test, validation split
X_train, y_train_prev, X_valid, y_valid_prev, X_test, y_test_prev = train_valid_test_split(df_upsampled,
                                                                          target = ['MultipleVariableDeclarations', 'AvoidReassigningParameters', 'ForLoo
                                                                          train_size=0.8,
                                                                          valid_size=0.1,
                                                                          test_size=0.1)

X_train = X_train['cleaned_code']
X_valid = X_valid['cleaned_code']
X_test = X_test['cleaned_code']
```

Python

Figure 16: Splitting and defining target variables

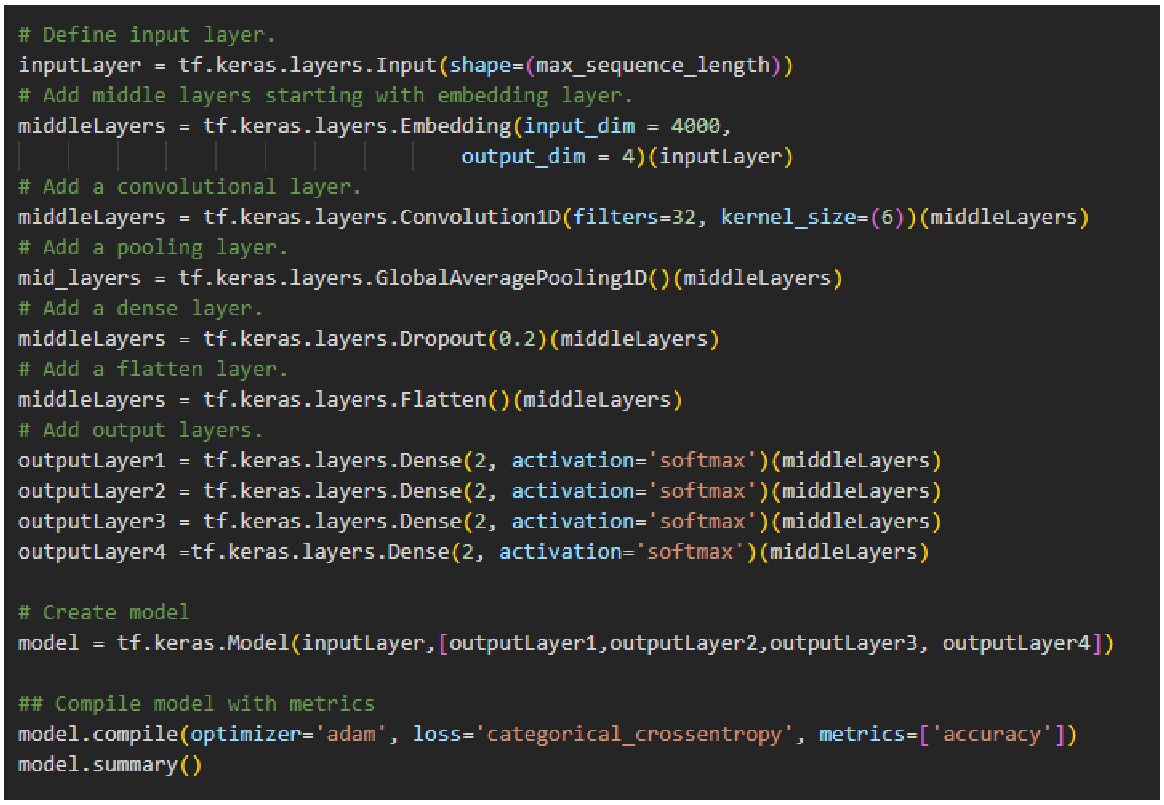
Tokenization was done because the input require consistent shape and length. This Python code is used to preprocess text data for use in a machine learning model. By converting the text into sequences of integers and padding these sequences, the text data is transformed into a format that can be used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



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

# Tokenize the text in X_train, X_valid, and X_test
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_valid_seq = tokenizer.texts_to_sequences(X_valid)
X_test_seq = tokenizer.texts_to_sequences(X_test)

max_sequence_length = 200
X_train = pad_sequences(X_train_seq, maxlen=max_sequence_length)
X_valid = pad_sequences(X_valid_seq, maxlen=max_sequence_length)
X_test = pad_sequences(X_test_seq, maxlen=max_sequence_length)

print('Number of tokens: ',len(tokenizer.word_counts))
sorted(tokenizer.word_counts.items(), key=lambda x:x[1], reverse=True)[0:10]
tokenizer.num_words = 1500
```

Figure 17: Tokenization

For this prototype, a CNN model was created with an input layer, embedding layer, a convolution layer, an average pooling layer, a dropout layer, a flatten layer and finally the 4 output layer binary split. Callbacks such as early stopping were defined to counter overfitting and the model was trained with 10 epochs and batch size 32.



```
# Define input layer.
inputLayer = tf.keras.layers.Input(shape=(max_sequence_length))
# Add middle layers starting with embedding layer.
middleLayers = tf.keras.layers.Embedding(input_dim = 4000,
                                    output_dim = 4)(inputLayer)
# Add a convolutional layer.
middleLayers = tf.keras.layers.Convolution1D(filters=32, kernel_size=(6))(middleLayers)
# Add a pooling layer.
mid_layers = tf.keras.layers.GlobalAveragePooling1D()(middleLayers)
# Add a dense layer.
middleLayers = tf.keras.layers.Dropout(0.2)(middleLayers)
# Add a flatten layer.
middleLayers = tf.keras.layers.Flatten()(middleLayers)
# Add output layers.
outputLayer1 = tf.keras.layers.Dense(2, activation='softmax')(middleLayers)
outputLayer2 = tf.keras.layers.Dense(2, activation='softmax')(middleLayers)
outputLayer3 = tf.keras.layers.Dense(2, activation='softmax')(middleLayers)
outputLayer4 =tf.keras.layers.Dense(2, activation='softmax')(middleLayers)

# Create model
model = tf.keras.Model(inputLayer,[outputLayer1,outputLayer2,outputLayer3, outputLayer4])

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

Figure 18: Building and Training Model

The entire implementation along with the thorough preprocessing steps and approach to solve data imbalance will be available on GitHub on a private repository and access will be given if requested.

7.4 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 08: TESTING

8.1 Chapter Overview

This chapter explores all the testing aspects for this project. Starting by rewinding through the goals and objectives set previously and moving forward with the determined testing criteria. Later, the model testing results, and the confusion matrix will be discussed and evaluated. Moving on to functional and nonfunctional requirements testing and mentioning any limitations faced.

8.2 Objectives and Goals of Testing

The ultimate aim for testing would be to adhere and make sure that your prototype functions and performs as expected. The testing objectives and goals for QualityCheck AI is mentioned below:

- To make sure that all components and modules of the QualityCheck AI system are working as expecting producing accurate results.
- Ensure that the functional requirements mentioned in the SRS with priority levels ("Must Have" and "Should Have") according to the MoSCoW principle is met.
- Ensure that the non-functional requirements mentioned in the SRS with priority levels ("Must Have" and "Should Have") according to the MoSCoW principle is met.
- Document or fix any errors/fixes in the system through testing.
- Carry out steps and testing to make it a benchmark and produce this system as a baseline for future researches in this domain.

8.3 Testing Criteria

In order to categorize the testing criterion, two particular ways have been distinguished:

1. **Functional Testing**
   This is where the focus is on the system meeting its intended use which are the functional requirements and how well the systems perform in achieving these.
2. **Non-functional Testing**

This is where the focus is on the system meetings its structural and non-functional use cases and how well the system performs in achieving these.

8.4 Model Testing

The dataset was split into training, test and validation test in the standard ration of 80:10:10 where all the records went through data preprocessing and tokenization before training the model.

8.4.1 Confusion matrix

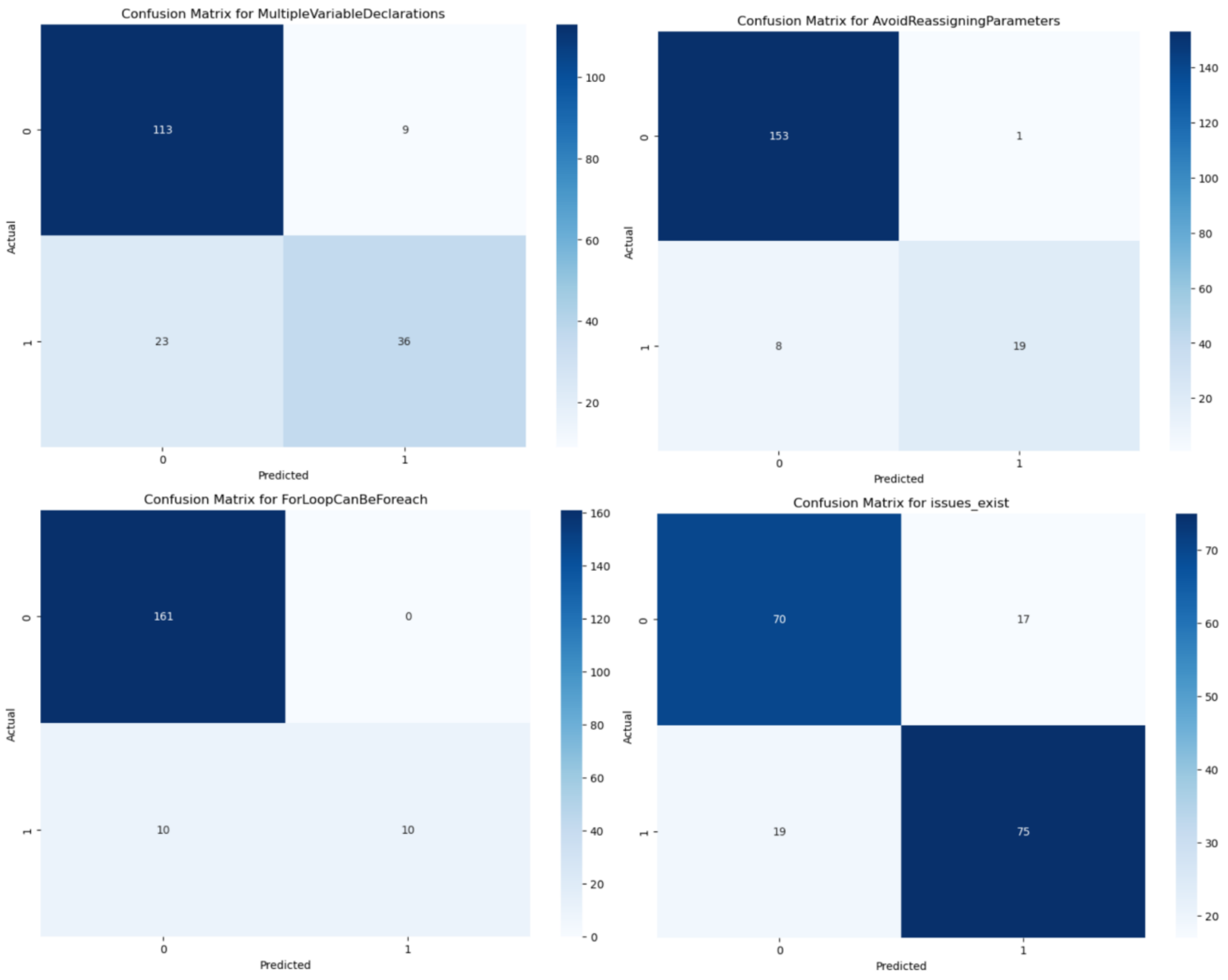


Figure 19: Confusion Matrix for each Code Quality Issue

Overall, the module performs well in terms of classification. Even with ain imbalanecd dataset, the model seemed to accurately classifiy most of the cases. The model performs the best for MultipleVariableDeclarations issue, AvoidReassigningParameters issue and for classyfing no issues in the code with less false positves and false negative making it a very reliable prediction. However, the ForLoopCanBeForEach issue is not as strong but is still considered decent considering the other evaluation metrics discussed below. Overall, it is a good benchmark produced and a novel identification where other reseraches could build on top of and comapre with.

8.4.2 Evaluation metrics

Evaluation Metric / Code Quality Issue	Accuracy	Precision	Recall	F1 Score
MultipleVariableDeclarations	0.82	0.80	0.61	0.69
AvoidReassigningParameters	0.95	0.95	0.70	0.81
ForLoopCanBeForEach	0.94	1.0	0.50	0.67
No Issues	0.80	0.82	0.80	0.81

Also, the no issues classification which had an accuracy of 0.80, precision of 0.82, recall of 0.8 and f1 score of 0.81 was obtained successfully proving the success of detecting code quality issues in AI-generated code. This is an extra classification used to get the confidence level.

8.5 Benchmarking

As previously discussed during the literature review, the author is the first to explore code quality issues in AI-generated code. Therefore, there is no standard benchmarking dataset to compare findings. In fact, there is not much data or datasets available for this domain at all.

8.6 Functional Testing

Test Case	FR ID	Action	Expected Result	Actual Result	Status
1	FR03	Type or paste Java code to test for code quality issues	Given AI-generated code visible in the text area box	Given code successfully displayed in text area box	Pass
2	FR01, FR02	User predicts for issues clicking 'Analyze' button	System predicts and results displayed with probabilities of identified issues.	System predicts and results displayed with probabilities of identified issues.	Pass
3	FR04	Check for analysis results for context	GUI displays the types of issues identified in a meaningful way	GUI displays the types of issues identified in a meaningful way	Pass
4	FR05	Select different versions of the models and predict an analysis	System uses selected model to predict and display results specific to that model	System displays analysis results specific to that model	Pass
5	FR06	Try clicking analyze without inputting any code	System does not allow to proceed and asks user to enter code to proceed	System does not allow to proceed and asks user to enter code to proceed	Pass
6	FR01, FR02	Try inputting code with specific code quality issue in it	System accurately predicts and identify the specific code quality issue and display in meaningful way	System accurately predicts and identify the specific code quality issue and display in meaningful way	Pass

8.7 Module and Integration Testing

Module	Input	Expected Result	Actual Result	Status
Tokenizer Module	Java code block	Code tokenized into a sequence or list of tokens	Code tokenized into a sequence or list of tokens	Pass
Code Quality Issue Detection Module	Java code block with code quality issues	Code quality issue identified along with confidence	Code quality issue identified along with confidence	Pass

8.8 Non-Functional Testing

8.8.1 Accuracy Testing

Covered in section *8.4.2* where an average of 90.3% accuracy was achieved for the 3 types of code quality issues.

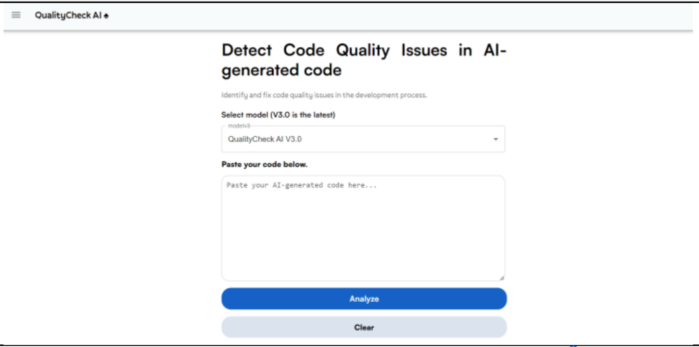
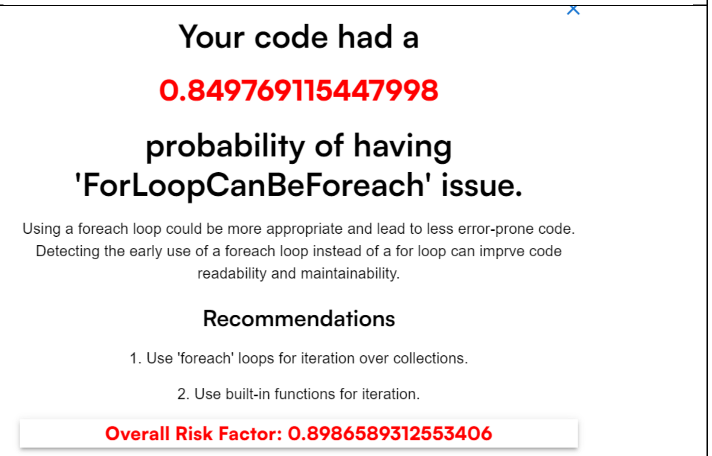
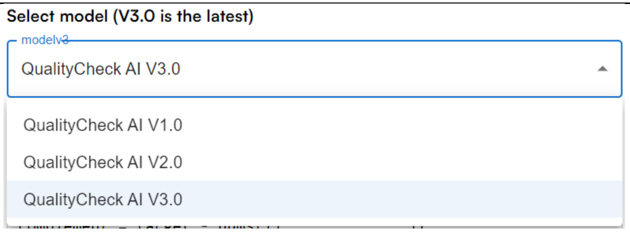
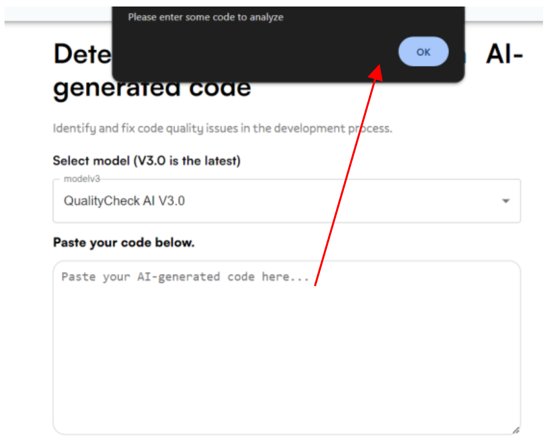
8.8.2 Performance Testing

In exhaustive testing of the prediction duration, it was noted that all predictions took under 1s despite the length of the code block inputted. This makes it scalable and efficient.

8.8.3 Reliability Testing

The reliability of the results and cliassfications can be proven by the reeulst of the confusion matrix. A breakdown and discussion was explained in section *8.4.1*

8.8.4 UI/UX Testing

Metric	Explanation	Screenshot
Engagement	Simple and user-friendly UI which the user finds easy to understand and engage	 QualityCheck AI Detect Code Quality Issues in AI-generated code Identify and fix code quality issues in the development process. Select model (V3.0 is the latest) modelv3: QualityCheck AI V3.0 Paste your code below. Paste your AI-generated code here... Analyze Clear
Performance	Time taken for prediction always under 1s therefore the making it efficient	 Your code had a 0.849769115447998 probability of having 'ForLoopCanBeForeach' issue. Using a foreach loop could be more appropriate and lead to less error-prone code. Detecting the early use of a foreach loop instead of a for loop can imprve code readability and maintainability. Recommendations 1. Use 'foreach' loops for iteration over collections. 2. Use built-in functions for iteration. Overall Risk Factor: 0.8986589312553406
Flexibility	The freedom for the user to choose preferred model according to their own accords.	 Select model (V3.0 is the latest) modelv3: QualityCheck AI V3.0 QualityCheck AI V1.0 QualityCheck AI V2.0 QualityCheck AI V3.0
Error handling	Meaningful error messages to guide the user on user preventable mistakes	 Please enter some code to analyze OK Dete AI- generated code Identify and fix code quality issues in the development process. Select model (V3.0 is the latest) modelv3: QualityCheck AI V3.0 Paste your code below. Paste your AI-generated code here...

8.9 Limitation of the Testing Process

No standard benchmark: since this was the first project of its nature targeting AI-generated code, there was not standard benchmark dataset or results to compare against.

No standard dataset and limited resources: since the field and rise of code generation tools are new, there were not much data readily available. Also, resources and understanding of deep learning applied in this less. Making it a less resourceful area in context of deep learning and code quality issues.

8.10 Chapter Summary

This covered all the testing aspects for this project. The goals and objectives set previously were recapped along with the determined testing criteria. Later, the model testing results, and the confusion matrix were be discussed and evaluated. Then the functional and nonfunctional requirements testing were covered mentioning any limitations faced.

CHAPTER 09: EVALUATION

9.1 Chapter Overview

This chapter focuses on a structured evaluation of the system that has been developed – Detecting Code Quality Issues from AI – Generated code. The areas under this section include the Methodology, Evaluation Criteria then moving on to the strict self-evaluation, selection of the evaluators and the results of it. The limitations of the **e**valuation is what the **c**hapter would be ended with.

9.2 Evaluation Methodology and Approach

The plan used both qualitative and quantitative ways. Quantitative looks at numbers like Accuracy, Precision, Recall and the F1 score to judge performance. On the other hand, qualitative includes expert reviews, user interviews and surveys. This covers looking at both how well the model works and how users feel.

9.3 Evaluation Criteria

Selected evaluation criteria and their purposes were discussed in the table below.

Criteria	**Purpose**
Innovation	Assess the research gap that was addressed and the technology used to meet it
Complexity and Scope	Evaluate from end to end of the scope of research and the problems that can arise
Technology Suitability	Evaluate whether the correct tools, technologies and practices have been applied
Code Quality Assessment	Evaluate how accurate and reliable the output of the model is
Performance Metrics	Evaluate the relevance and effectiveness of the performance metrics that have been selected
User Interface and Experience	Evaluate Usability and the Satisfaction of the User with the interface that has been developed from end to end
Feedback and Future Direction	Evaluating the feedback that was given and fostering it for future development

Table 17: Evaluation Criteria

9.4 Self Evaluation

Self-evaluation of **QualityCheck AI** has been discussed in the table below

Criteria	**Evaluation**
Novelty	Software development faces issues with the go to methods of detecting code quality issues, with the AI driven approach of this model, it aims to bridge that gap and acts as mitigation for the code quality issues
Challenge and Scope	The domain that is being explored proves challenging due to the limited exploration in this area. The challenges also include the inherent issues with pre-written programming templates and the lack of labeled data is also one to consider.
Technology Selection and Development Approaches	The model has been built using the knowledge gathered from the industry and industry standard best practices in development. However, the ever-evolving technologies and expert evaluation needs to be considered as well
Code Quality Assessment	The model shows promise in terms of identifying code quality issues and detecting anomalies, the issues to be considered stem from the lack of labeled data.
Performance Metrics	The model has good performance showing high accuracy rates, good classification capability and minimizing false positives and negatives to ensure reliability.

Table 18: Self Evaluation

9.5 Selection of the Evaluators

The evaluators of this project are categorized as displayed in the table below.

Category ID	**Description**
CAT1	Domain experts such as Quality Assurance experts and senior developers in the industry, who endorse the need

	for secure systems and checks for quality and review code.
CAT2	Technical Experts including graduates, researchers, and lecturers - with knowledge in related areas like machine learning, data engineering, and software were also consulted.
CAT3	Targeted Users – Developers, students and QA engineers were considered.

Table 20: Evaluator Categories

9.6 Evaluation Result

The thematic analysis findings for each criterion and categories were presented in the table below.

Criteria	Category ID	Theme	Summarized Opinion
Novelty	CAT1, CAT2	Originality in the selected problem domain and what was conducted as being innovation in research	There is originality as it deals with problems of missing labeled data and using deep learning for identifying code quality issues of AI generated code. Also using 4 binary output stream CNN sounds like a novel contribution.
Challenge and Scope	CAT1, CAT2	Overcoming obstacles and defining project boundaries	Main hardships faced include the lack of data for deep learning models. Also, since it is a ever evolving field, there are not many research material on this domain and approach.
Code Quality Issues Detection	CAT1	Severity of code quality issues and system accuracy	The code quality issues selected are adequate and some avoidable issues, the model has detected majority of the issues

and Range			in AI generated code but certain issues still remain in AI-generated code.
Technology stack and approach	CAT2	Technology stack choice and coding methodology	To ensure efficiency and scalability, the research project used the best possible technology stack, utilizing React for frontend development and Python Flask for backend development. Version control was managed through Git and for the scope of this project its viable.
Performance Metrics	CAT2	Evaluation metrics analysis and proficiency	There have been performance metrics that have been generated for the model to get an idea of the models performance which include the confusion matrix, accuracy, recall, precision and F1 score which is the ideal stack for this kind of problem.
UI/UX	CAT3	User interface and user experience	Users can access previous models and choose. The interface has simple and intuitive design. Even though input fields are meant to prevent errors, some evaluators thought there could be more to the application.
Future work	CAT1,	Future work and feedback	Issues that can be considered to

and feedback	CAT2, CAT3		be overcome is the dataset imbalance that can improve the model performance, turning the model into an IDE plugin can be looked at in the long run or even multi language support as well.

Table 21: Results of Evaluation (Thematic)

9.7 Limitations of Evaluation

One big problem that arose during the thesis project's evaluation was the difficulty in finding specialists who had firsthand knowledge of projects close to the thesis. A number of experts in directly relevant fields were contacted but were unable to take part because of their busy schedul es, that resulted in having a smaller number of evaluators for that project.

9.8 Evaluation of Functional Requirements and Non-Functional Requirements

Section 8.6 provides a thorough evaluation of the functional requirements, and Section 8.8 assesses the non-functional requirements. The results of the analysis demonstrated that every requirement had been successfully implemented in accordance with the suggested plan, guaranteeing the project's successful completion of its functional and non-functional components.

9.9 Chapter Summary

This chapter offers a thorough analysis of the created model, outlining the methods and standards used in the evaluation process. It comprises the evaluation of the author as well as the classification and display of the reviewer's comments. The chapter also describes the evaluation process's constraints and provides advice on where to find the functional and non-functional requirements needed for additional research and development.

CHAPTER 10: CONCLUSION

10.1 Chapter Overview

This chapter concludes the thesis and highlights 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.

10.2 Achievements of Research Aims & Objectives

Research Objectives	**Learning Outcome**	**Status**
Problem Identification	LO1, LO2	Completed
Literature	LO1, LO4, LO8	Completed
Data Gathering and Analysis	LO2,LO3	Completed
Research Design	LO1,LO5, LO8	Completed
Implementation	LO1,LO5,LO7,LO8	Completed
Testing and Evaluation	LO5,LO8	Completed

Table 22: Achievements of Research Aims & Objectives

10.3 Utilization of Knowledge from the Course

Module	Utilized Knowledge
Programming Principles I, Programming Principles II, Object Oriented Programming	Simple coding to advanced object-oriented concepts is a necessary requirement to any developer which is always utilized wherever and whenever applicable.
Applied Artificial Intelligence, Machine Learning and Data Mining	The algorithmic knowledge gained from this module was very valuable. Especially learning the fundamentals of machine learning and basics of deep learning helped as a stepping stone.
Software Development Group Project (SDGP)	This module provided a reality into the industry. Research principles and best practices gained in this played a crucial role. From documentation to writing code whether it be for the frontend and backend in high professional standard
Web Design and Development	An introduction to creating a display page and the entire web development process helped understand the necessary fundamentals required for a frontend.
Usability Testing	This module gave a strong understanding on how to design and the best practices to consider when designing and developing an application in context to the user's experience.

Table 23: Utilization of Knowledge

10.4 Use of Existing Skills

The completion of the placement year at WSO2 as a software engineering intern gave the author an invaluable experience. The industry standard, best practices and supportive mentors made it extremely handy to complete this project on time.

Also, the author's research skills developed through leading the team during the SDGP module and working on R&D projects during the internship played a vital role in exploring the research domain and write proper reports.The DL and ML knowledge gained from the second year onwards helped understand and grasp the theoretical concepts of main algorithms and techniques used in ML and DL which increased understanding and played as a key asset.

10.5 Use of New Skills

- **Code Quality in Generative AI**: During the start of the project, the author lacked understanding in the domain of code quality and how generative AI works. Therefore, the author gained invaluable insights in this domain and where the future is headed with generative AI.

- **Deep Learning (CNN)**: The QualityCheck AI architecture and system uses a CNN with multiple different types of parameters and layer architectures. This made the author gain a deep understanding of CNNs and how to customize certain layers for specific problems.

10.6 Achievement of Learning Outcomes

Description	Learning Outcome(s)
A challenging project scope was defined and tackled using appropriate methodologies and techniques.	L01
The schedule was followed and updated in the form of a GANTT chart to meet milestones and deadlines.	L02
Relevant data gathering and requirement elicitation was conducted.	L03
LR, brainstorming and interviews were conducted to retrieve findings and information	L04
Literature review was constantly updated and reviewed till the end of the project to gain a thorough and latest understanding. Brainstorming and interviews were conducted to retrieve more findings.	L05, L07, L08
Social, Legal, Ethical and Professional issues were always monitored and ensured to stay	L06

within the boundaries of the research.	
Presentation and review for viva to publish the submitted thesis.	L09

Table 24: Achievement of Learning Outcomes

10.7 Problems and Challenges Faced

Challenge encountered	**Counter**
Data scarcity(imbalanced) and limited resources	No readily available data due to the newness of the generative AI domain. The data for the project had to be extracted from a Json file and be converted into processable format.
Hardware constraints	Assinging and using GPU resources were power consuming and needed more GPU. Virtual enviroenments such as Google Colab, Code Spaces by Github were used to counter and deal with heating issues.
Finding research materials	Since the author is the first to explore applying deep learning in this domain, there were no prior work in this domain. Similar work in other domains had to be compared and test if applicable.

Table 25: Challenges Encountered

10.8 Deviations

10.8.1 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.

10.8.2 Schedule Related Deviations

There are no schedule related deviations as of April 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	Completed
Testing and Evaluation	March 2024	Completed
Final Thesis Submission	April 2024	Completed

Table 26: Schedule Related Deviations

10.9 Limitation of the Research

Limitations identified for QualityCheck AI are discussed below:

- Model performance in classification achieved an overall good result achieving good scores in accuracy, precision, recall and f1 score. However, some code quality issues can improve its performance because some measures can be considered as not good.

- Dataset imbalance would be the cause for loss and potential poor performance in real world scenarios. Since it is a new field, there will always be a data scarcity and researches can use this as a benchmark or gather more data to get more accurate results.

- QualityCheck AI has been built only to analyze issues in Java source code which can be considered a limitation as it does not have multi language support.

10.10 Future Enhancements

- Gathering more data for code quality issues with less data or maybe even curating a new dataset which can be a standard for this domain. Having balanced data can lead to extraordinary results.
- Identifying more range of code quality issues which can be considered as a worthful aspect of this system.
- Include multi-language support. The existing system could be used to cater more programming languages or ensemble model techniques could be explored to implement multi language support.
- Frontend or UI could be included to have more features such as downloading the analysis as a report and other innovative features.

10.11 Achievement of the Contribution to Body of Knowledge

The contribution made to the body of knowledge in context to theory and technicality can be summarized as follows:

1. **Contribution to Deep Learning Domain (Research and Theoretical)**
   Prior work have only used basic deep learning mechanisms with sequential approaches and one output stream for normal human code snippets. One the other hand, the system by QualityCheck AI proposes a novel deep learning model which uses a 4 output stream binary split which accurately detects code quality issues have been implemented. Further, it has been built targeting AI-specific code.
2. **Contribution to Code Quality Domain (Technical and Problem)**
   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. The proposed system and test results with outstanding evaluation metric scores prove that QualityCheck AI can perform well and serve as a benchmark for future research in this domain. This helps improve current static analysis procedures and adapt static analysis and code quality security in terms of the future where its headed towards majority of the code being AI-generated.

10.12 Concluding Remarks

QualityCheck AI is a system which is the first of its kind. It proposes a novel deep learning approach of using a CNN with 4 output stream binary split for code analysis. Quality Check AI proves the ability of deep learning in context to code analysis and accurately detecting multiple code quality issues where it can be considered more effective than existing code analysis tools and techniques. This research project opens various possibilities in the emerging field of generative AI. Being the first research in applying deep learning for detecting code quality issues in AI-generated code, this could be considered a dawn for a new era by using this as a steppingstone.

References

Alkaoud, M. and Syed, M. (2020). On the Importance of Tokenization in Arabic Embedding Models. In: Zitouni, I. Abdul-Mageed, M. Bouamor, H. et al. (eds.). *Proceedings of the Fifth Arabic Natural Language Processing Workshop*. December 2020. Barcelona, Spain (Online): Association for Computational Linguistics, 119–129. Available from https://aclanthology.org/2020.wanlp-1.11 [Accessed 9 April 2024].

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.

Azeem, M.I. et al. (2019). Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, 108, 115–138. Available from https://doi.org/10.1016/j.infsof.2018.12.009.

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.

Chen, M. et al. (2021). Evaluating Large Language Models Trained on Code. *ArXiv*. Available from https://www.semanticscholar.org/paper/Evaluating-Large-Language-Models-Trained-on-Code-Chen-Tworek/acbdbf49f9bc3f151b93d9ca9a06009f4f6eb269 [Accessed 9 April 2024].

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].

Di Nucci, D. et al. (2018). Detecting code smells using machine learning techniques: Are we there yet? *2018 IEEE 25th International Conference on Software Analysis, Evolution and*

Reengineering (SANER). March 2018. Campobasso: IEEE, 612–621. Available from https://doi.org/10.1109/SANER.2018.8330266 [Accessed 9 April 2024].

Dwivedi, Y.K. et al. (2023). Opinion Paper: “So what if ChatGPT wrote it?” Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International Journal of Information Management*, 71, 102642. Available from https://doi.org/10.1016/j.ijinfomgt.2023.102642.

Eke, D.O. (2023). ChatGPT and the rise of generative AI: Threat to academic integrity? *Journal of Responsible Technology*, 13, 100060. Available from https://doi.org/10.1016/j.jrt.2023.100060.

Fan, C. et al. (2021). A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data. *Frontiers in Energy Research*, 9. Available from https://doi.org/10.3389/fenrg.2021.652801 [Accessed 9 April 2024].

Fang, W. et al. (2017). A Survey of Big Data Security and Privacy Preserving. *IETE Technical Review*, 34 (5), 544–560. Available from https://doi.org/10.1080/02564602.2016.1215269.

Fui-Hoon Nah, F. et al. (2023). Generative AI and ChatGPT: Applications, challenges, and AI-human collaboration. *Journal of Information Technology Case and Application Research*, 25 (3), 277–304. Available from https://doi.org/10.1080/15228053.2023.2233814.

Hamer, S., d’Amorim, M. and Williams, L. (2024). Just another copy and paste? Comparing the security vulnerabilities of ChatGPT generated code and StackOverflow answers. 22 March 2024. Available from https://www.semanticscholar.org/paper/29d8d9ef1f8a4c6a00efad51babdfff556738d1c [Accessed 3 April 2024].

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].

Khaire, U.M. and Dhanalakshmi, R. (2022). Stability of feature selection algorithm: A review. *Journal of King Saud University - Computer and Information Sciences*, 34 (4), 1060–1073. Available from https://doi.org/10.1016/j.jksuci.2019.06.012.

Lawlor, P. and Chang, J. (2024). The rise of generative AI: A timeline of breakthrough innovations. Available from https://www.qualcomm.com/news/onq/2024/02/the-rise-of-generative-ai-timeline-of-breakthrough-innovations [Accessed 8 April 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].

Lock, S. (2022). What is AI chatbot phenomenon ChatGPT and could it replace humans? *The Guardian,* 5 December. Available from https://www.theguardian.com/technology/2022/dec/05/what-is-ai-chatbot-phenomenon-chatgpt-and-could-it-replace-humans [Accessed 8 April 2024].

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].

Moha, N. et al. (2010). DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering*, 36, 20–36. Available from https://doi.org/10.1109/TSE.2009.50.

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].

Murugesan, S. and Cherukuri, A.K. (2023). The Rise of Generative Artificial Intelligence and Its Impact on Education: The Promises and Perils. *Computer*, 56 (5), 116–121. Available from https://doi.org/10.1109/MC.2023.3253292.

Ntoutsi, E. et al. (2020). Bias in data-driven artificial intelligence systems—An introductory survey. *WIREs Data Mining and Knowledge Discovery*, 10 (3), e1356. Available from https://doi.org/10.1002/widm.1356.

Pantiuchina, J., Lanza, M. and Bavota, G. (2018). Improving Code: The (Mis) Perception of Quality Metrics. *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. September 2018. 80–91. Available from https://doi.org/10.1109/ICSME.2018.00017 [Accessed 9 April 2024].

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].

Porter, J. (2023). ChatGPT bug temporarily exposes AI chat histories to other users. *The Verge*. Available from https://www.theverge.com/2023/3/21/23649806/chatgpt-chat-histories-bug-exposed-disabled-outage [Accessed 8 April 2024].

Probst, P., Boulesteix, A.-L. and Bischl, B. (2019). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *Journal of Machine Learning Research*, 20 (53), 1–32.

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].

Scharth, M. (2022). The ChatGPT chatbot is blowing people away with its writing skills. An expert explains why it’s so impressive. *The Conversation*. Available from http://theconversation.com/the-chatgpt-chatbot-is-blowing-people-away-with-its-writing-skills-an-expert-explains-why-its-so-impressive-195908 [Accessed 8 April 2024].

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.

Siddiq, M.L. et al. (2024). Quality Assessment of ChatGPT Generated Code and their Use by Developers.

Singh, A. (2024). 27 Amazing Generative AI Statistics (2024) You Should Know! *TechTipsWithTea*. Available from https://techtipswithtea.com/data/generative-ai-statistics/ [Accessed 8 April 2024].

Staff, I.S., GitHub. (2023). Survey reveals AI’s impact on the developer experience. *The GitHub Blog*. Available from https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/ [Accessed 3 April 2024].

Sultanow, E. et al. (2018). Machine Learning based Static Code Analysis for Software Quality Assurance. *2018 Thirteenth International Conference on Digital Information Management (ICDIM)*. September 2018. Berlin, Germany: IEEE, 156–161. Available from https://doi.org/10.1109/ICDIM.2018.8847079 [Accessed 9 April 2024].

Susarla, A. et al. (2023). The Janus Effect of Generative AI: Charting the Path for Responsible Conduct of Scholarly Activities in Information Systems. *Information Systems Research*, 34 (2), 399–408. Available from https://doi.org/10.1287/isre.2023.ed.v34.n2.

Taeb, M., Chi, H. and Bernadin, S. (2024). Assessing the Effectiveness and Security Implications of AI Code Generators. *Journal of The Colloquium for Information Systems Security Education*, 11 (1), 6–6. Available from https://doi.org/10.53735/cisse.v11i1.180.

Thorp, H.H. (2023). ChatGPT is fun, but not an author. *Science*, 379 (6630), 313–313. Available from https://doi.org/10.1126/science.adg7879.

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.

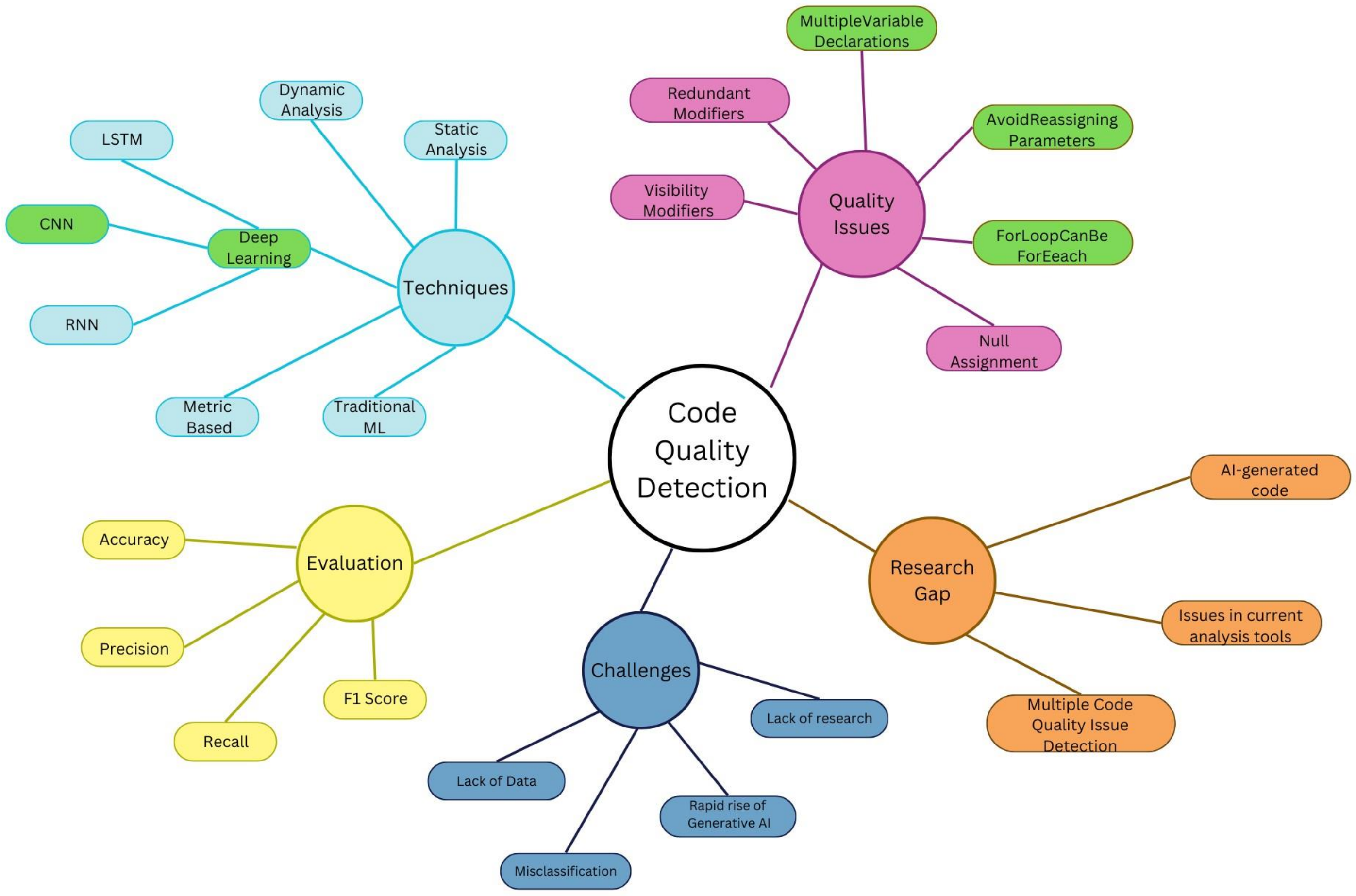
Vassallo, C. et al. (2020). How developers engage with static analysis tools in different contexts. *Empirical Software Engineering*, 25 (2), 1419–1457. Available from https://doi.org/10.1007/s10664-019-09750-5.

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].

Zhuo, T.Y. et al. (2023). Red teaming ChatGPT via Jailbreaking: Bias, Robustness, Reliability and Toxicity. Available from https://doi.org/10.48550/arXiv.2301.12867 [Accessed 8 April 2024].

Appendix A – Concept Map



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