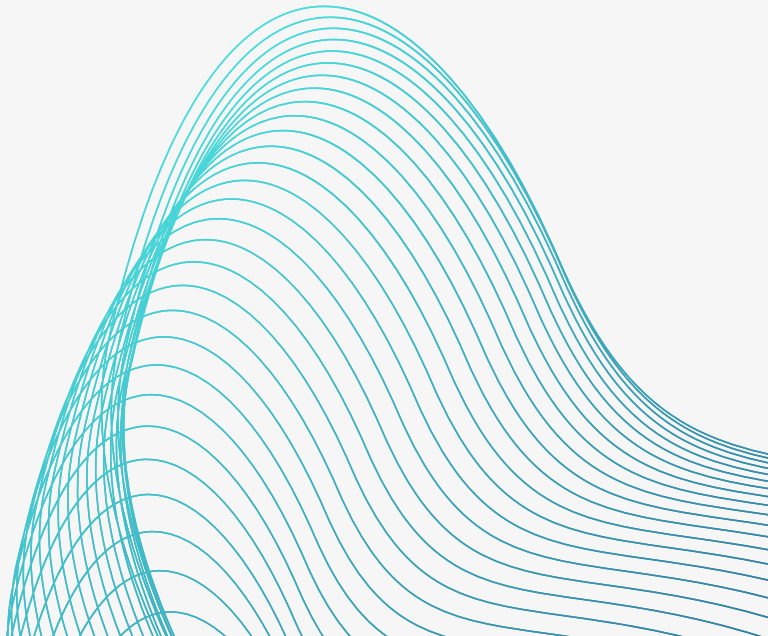
Created by
Dominators

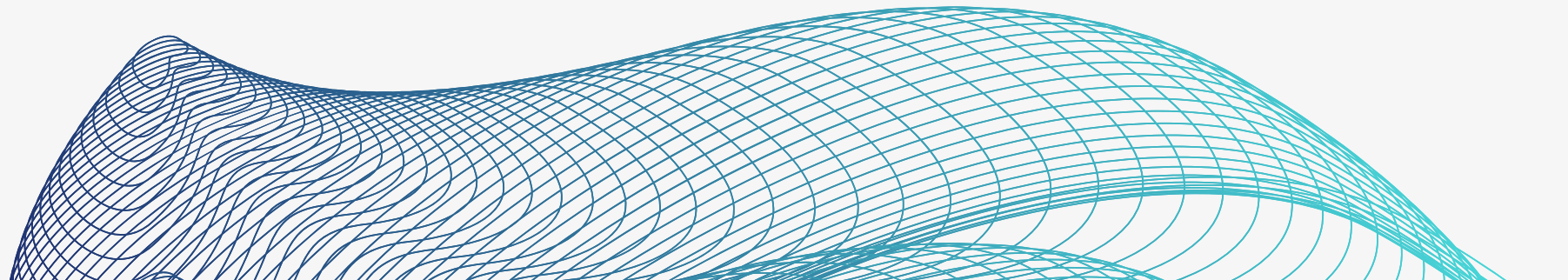
Automated Feedback Generation System for Introductory Programming Assignments

Supervisors:
Prof. (Mrs.) Thanuja C. Sandanayake
Dr. Supunmali Ahangama



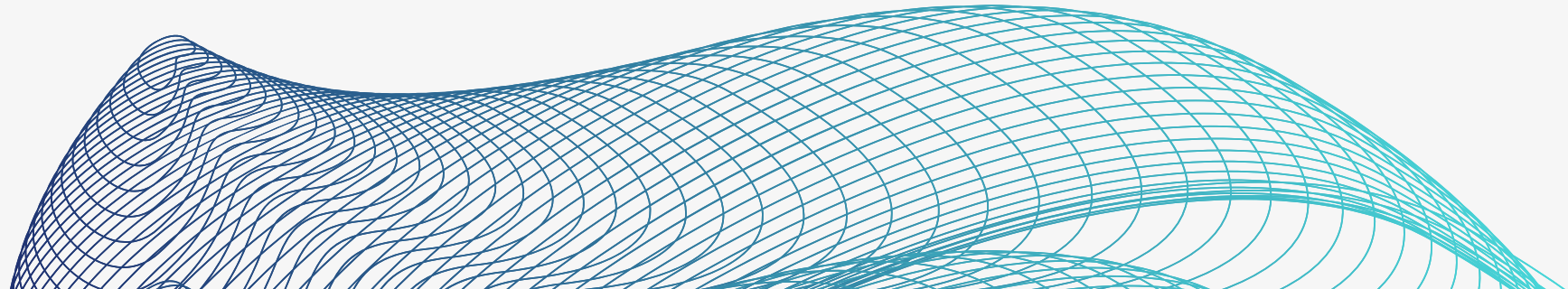
Our Team Members

195051E	Mathusha V.
195062M	Pavithra K.
195078R	Sathsarani W.H.G
195053L	Mithuja G.



Outline

- Introduction
- Problem in Brief
- Research Aim & Objectives
- Proposed Solution
- Project Approach
- Project Requirements



INTRODUCTION

- Programming is a fundamental and pivotal subject within the majority of Information Technology degree programs. Consequently, students are required to complete programming assignments as part of their coursework.

- The most basic form of automated feedback to programming measures whether the behaviour of students' code is as expected.

- These tools are often referred to as auto graders because they return a grade based on the student's code performance during testing. our work explores the extension of an auto grading system to design alternate grading schemes by using test cases results patterns to identify the concepts and skills students are struggling with, and grade based on these concepts and skills rather than based on the testing outcomes.

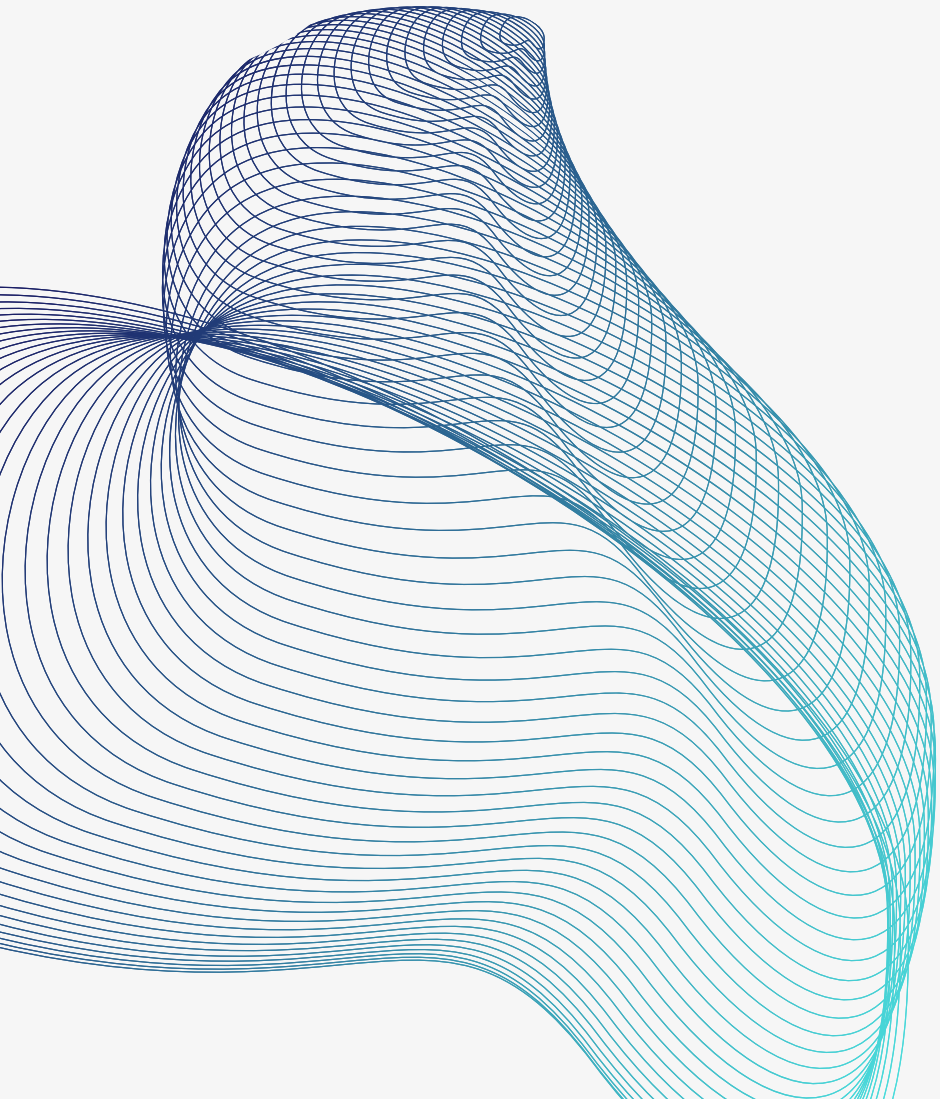
- This system first define the corrections to errors that students might make, Repair the Programme and give the corresponding minimum fixed and rewrite the program ,after that analysis the code and show the quality of the code, after that Analysis the Code Vulnerability for ensuring code security.

Problem in Brief

- The absence of prompt feedback prevents students from understanding their strengths and weaknesses and hampers their learning process. Additionally, instructors face challenges in providing timely and meaningful feedback due to manual grading, resource constraints, and scalability issues

- This lack of timely feedback impedes students' ability to correct their mistakes promptly and gain a deeper understanding of programming. Overall, the problem affects student learning, skill development, and the instructor-student relationship, emphasizing the need for efficient feedback mechanisms.

RESEARCH AIM



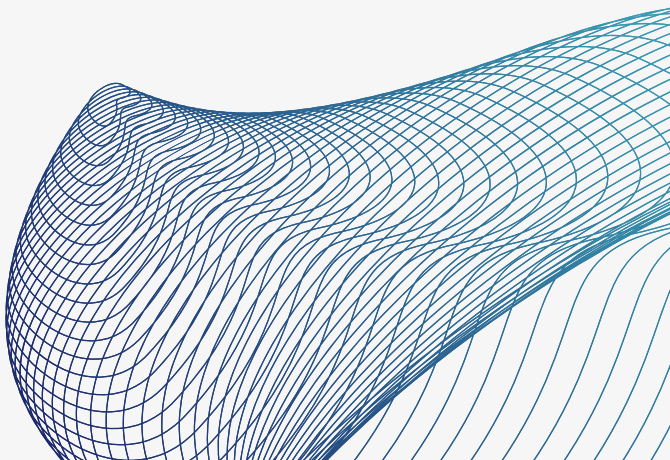
To develop system to provide automated personalized feedback for the learners in Fundamentals of programming language assesment.

RESEARCH OBJECTIVES

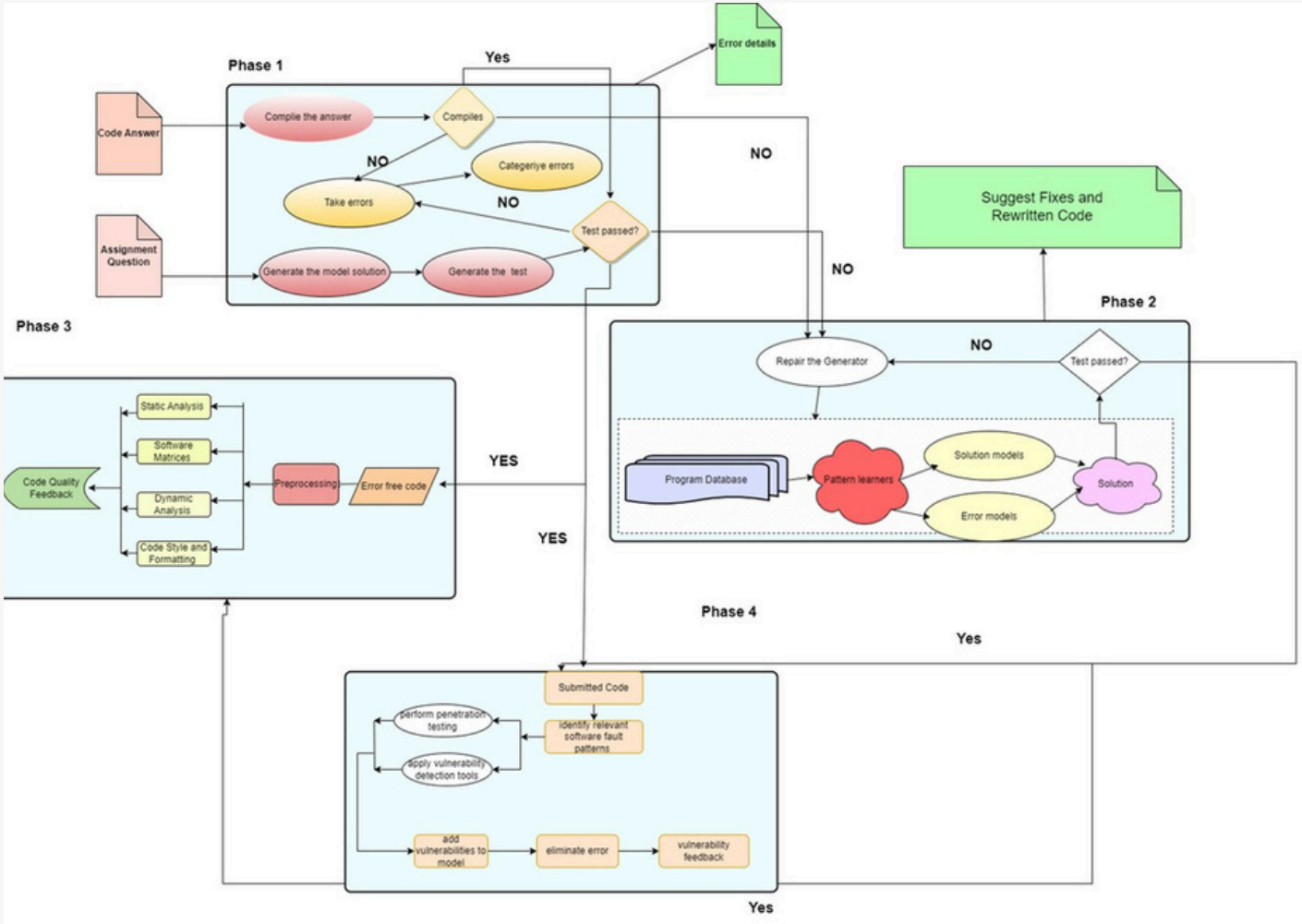
- To develop a module to detect and classify the error of the program
- To develop a module to Repair the Programme and corresponding minimum fixed and rewrite the program
- To develop a module to Analysis the Code Quality or Style Feedback
- To develop a module to Analysis the Code Vulnerability

Proposed Solution

we present an innovative feedback generation system that aims to enhance student efficiency by providing timely and constructive feedback. Our system consists of four modules, each addressing a specific aspect of the feedback generation process.



System Architecture



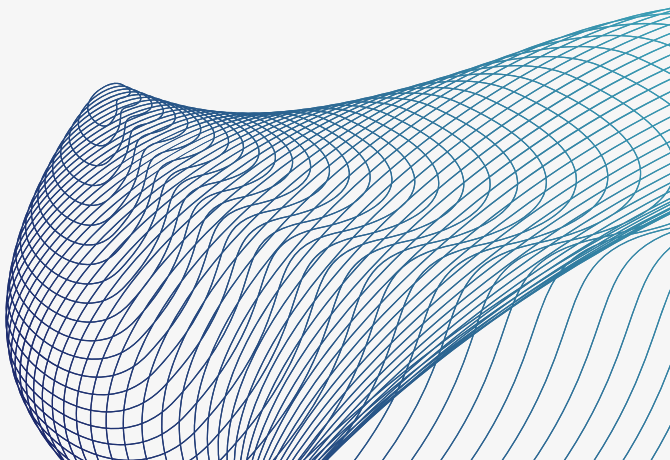
Module 01:

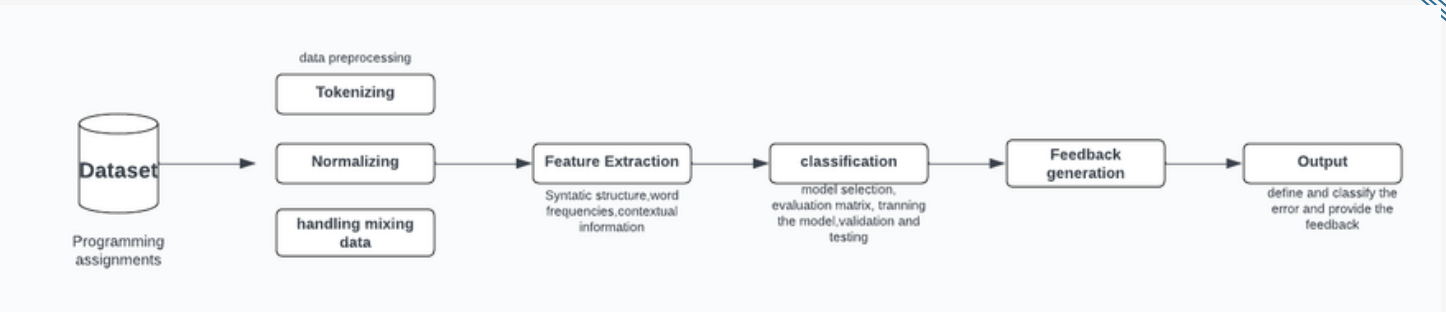
Classify or Detect the Error

This module Focus on automatically identify and categorise errors within code or data

Objective: to detect and classify errors in Java programming assignments and provide incremental feedback to students.

Benefit: pinpointing common issues and tailoring their instruction accordingly.





Preprocessing : preprocess the dataset means removing any irrelevant or noisy data, such as blank lines or comments. also includes tokenizing it means the code breaking it up into individual words or phrases.

Classification : classify the code into different error categories

Feedback Generation: Generating specific, actionable feedback for each identified error, guiding students on how to correct their mistakes.

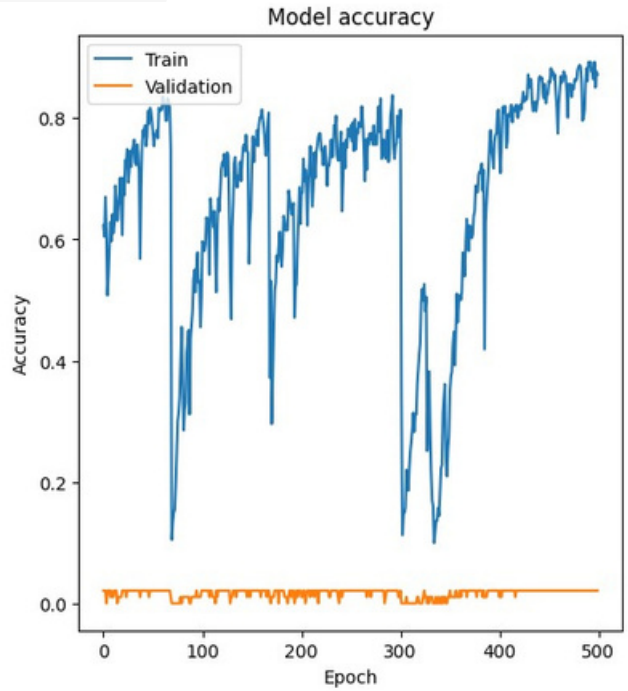
Research Gap

- Timely and effective formative feedback, the challenges faced by Intelligent Tutoring Systems (ITS), and the limitations of current automated programming error feedback approaches

- Existing automated feedback systems primarily focus on binary feedback or generic error messages that do not consider the specific context of the student's code. This lack of context-awareness can lead to misunderstandings and hinder learning progress.

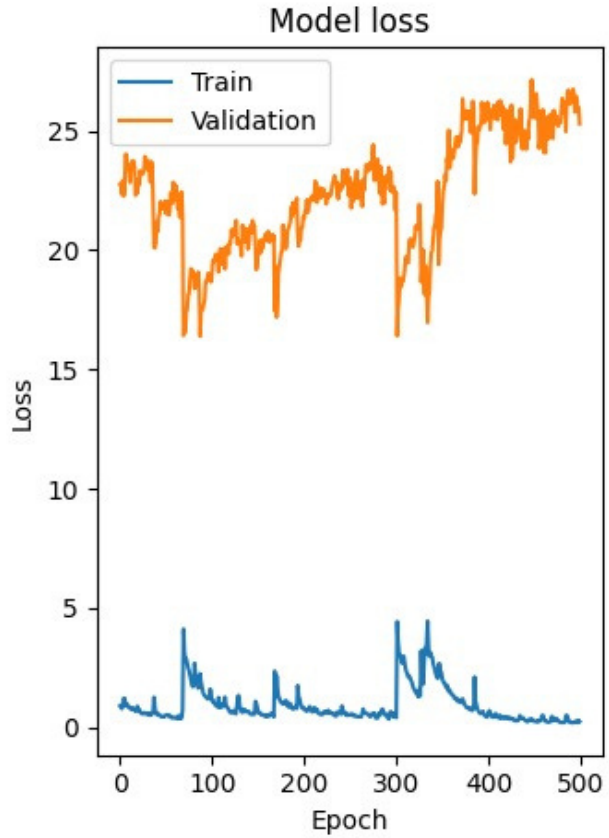
- Introducing a novel feature that pinpoints specific errors on any platform, enabling students to identify mistakes precisely and receive targeted feedback for efficient correction.

Evalution

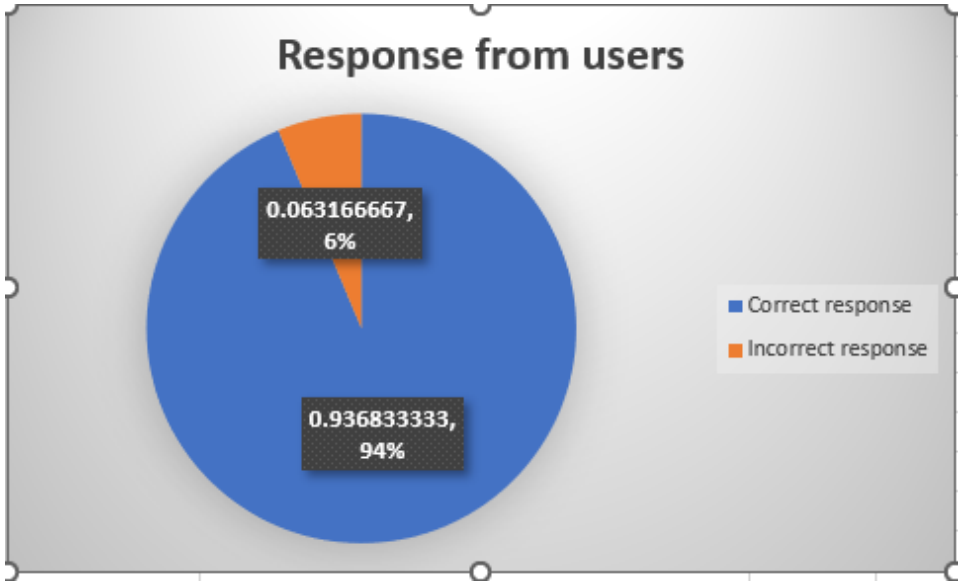


Model Accuracy

Model Loss



Manual Evaluation

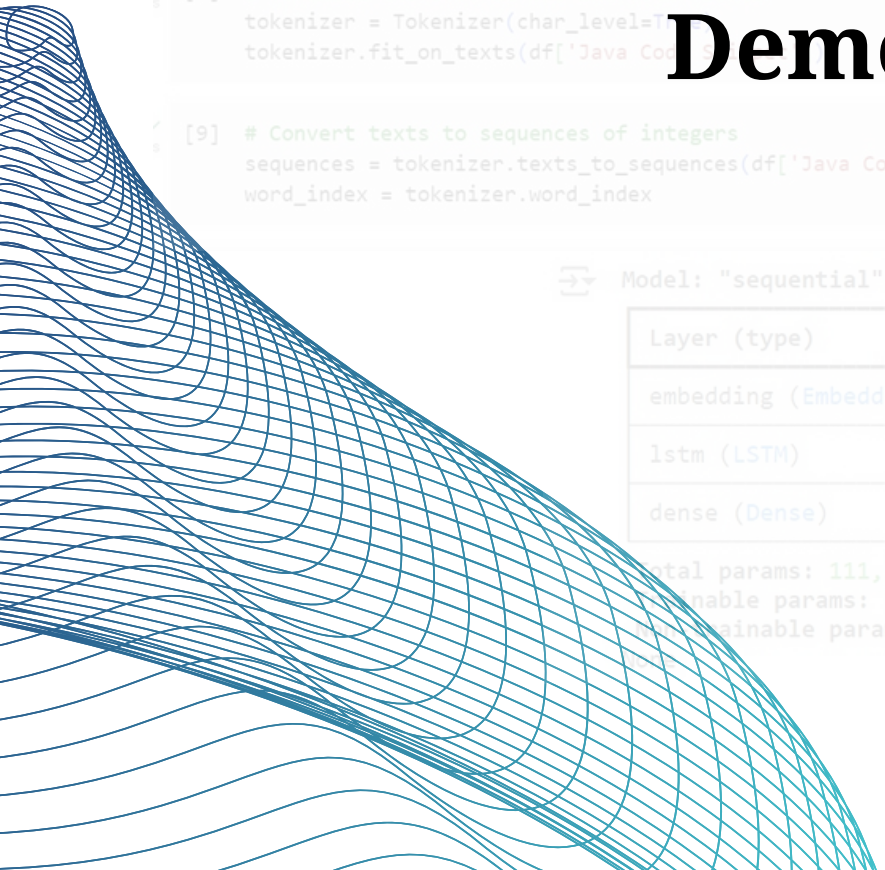


validate the system's performance, I conducted a manual evaluation involving user feedback

- **Reviewing the system's outputs.**
- **Checking if the outputs are correct or incorrect based on the expected results.**
- **Collecting user feedback to determine if the outputs are correct or incorrect.**

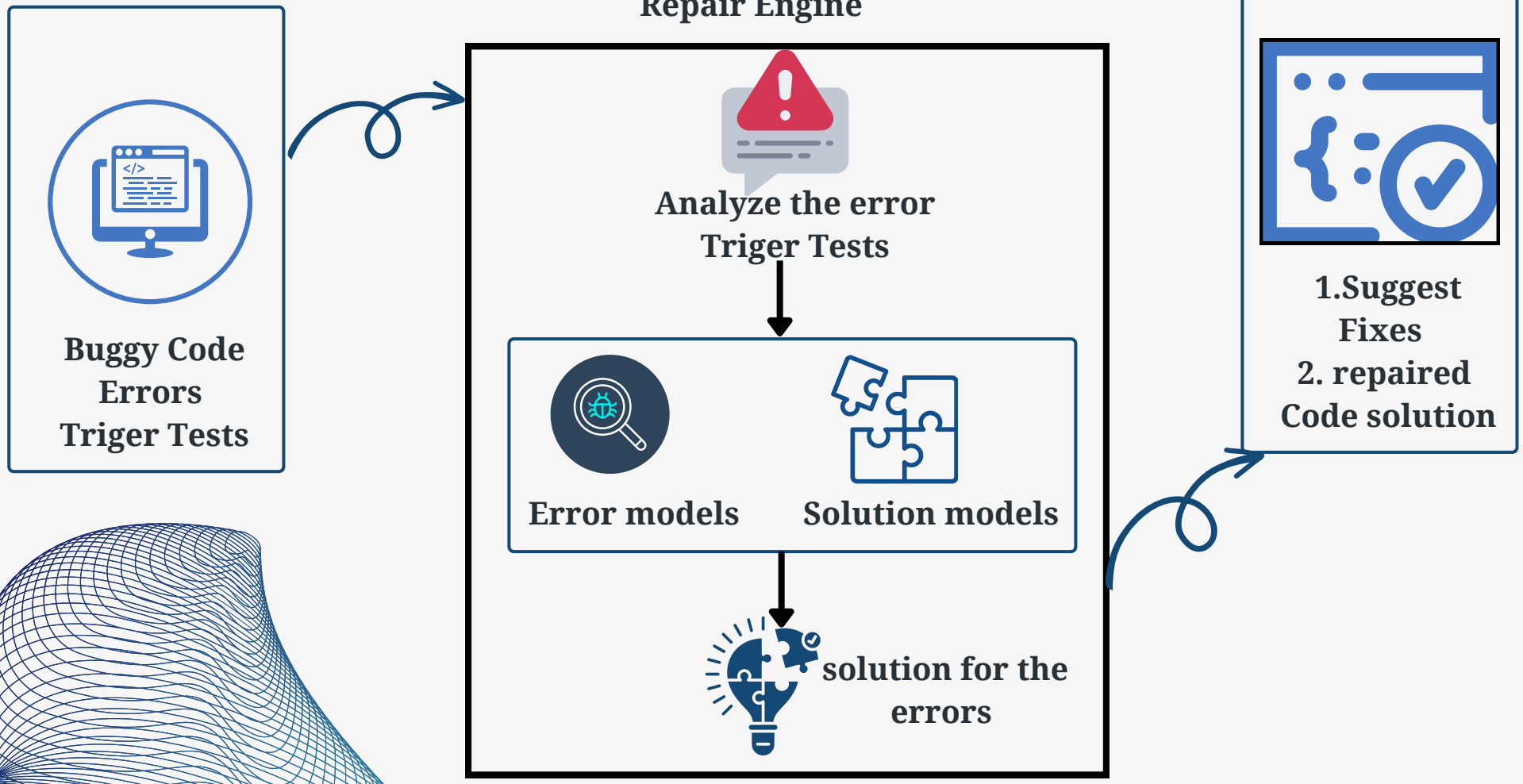
This process ensures the accuracy and reliability of the system in detecting and classifying errors based on real user responses.

Demonstration



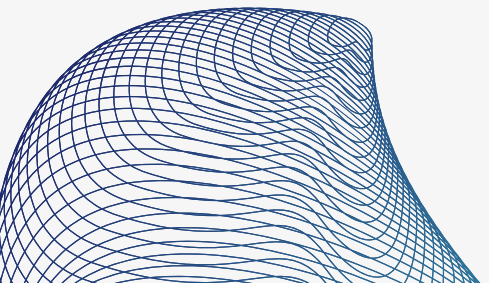
Module 2:

Repair the Program



RESEARCH GAP

- **Insufficient Research**: Lack of comprehensive study on LLM-based function-level APR.
- **Performance Loss**: Significant performance drop in function-level APR.
- **Few-Shot Learning**: Effectiveness in function-level APR not well-validated.
- **Auxiliary Information**: Underexplored use of bug reports and tests for repair.
- **Costly Techniques**: Need for more cost-effective function-level fault localization.
- **Limited Techniques**: Need for practical LLM-based function-level APR methods.

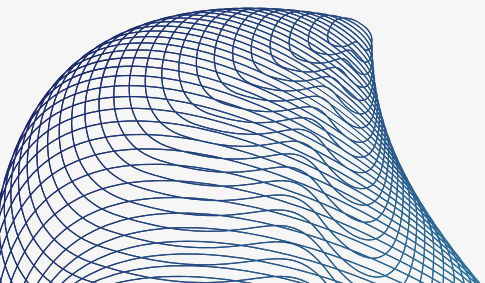


SOLUTION IN DETAIL (CONTD.)

Repair the Program

- The core of Module 2 is the "**Suggest Fixes and Rewritten Code**" component. It suggests specific corrections for the code, provides guidance on making these corrections, and, in some cases, offers entirely rewritten code segments to illustrate the necessary changes.

- The "**Repair Engine**" is responsible for generating the suggested fixes, guidance, and rewritten code. It uses few-shot learning mechanism and logics to determine how the code should be corrected and improved



REPAIR MODULE OVERALL MODELS ARCHITECTURE

Suggestion Model

Buggy Code
Trigger Test
Error Message

GPT 3.5 Turbo

Repair Suggestions

Repair Suggestion 1

Repair Suggestion 2

Repair Suggestion 3

Patch Generation Model

Buggy Code

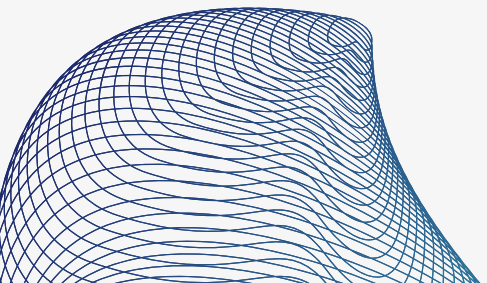
Repair Suggestion

open-source
Magicoder model

Fixed Code

Fixed Code

Fixed Code



EVALUTION

Data Set Format

Dataset	Project	# Bugs	SF Bugs
Defects4j 1.2	Chart	25	16
	Closure	140	105
	Lang	56	42
	Math	102	74
	Mockito	30	24
	Time	22	16
Defects4j 2.0	Cli	30	28
	Codec	13	11
	Collections	2	1
	Compress	40	36
	Csv	13	12
	Gson	12	9
	JacksonCore	18	13
	JacksonDatabind	85	67
	JacksonXml	5	5
	Jsoup	58	53
	JxPath	14	10
Overall		665	522

MODELS EVALUTION

SF Bugs Plausible Fixes

	GPT-3.5-Turbo	Coddegen	Our Repair module
Chart	12	11	14
Closure	40	30	56
Lang	19	25	32
Math	48	43	55
Mockito	8	8	12
Time	7	5	7
Cli	16	13	19
Codec	8	5	11
Collections	0	1	1
Compress	21	22	28
Csv	10	9	11
Gson	6	8	9
JacksonCore	9	6	10
JacksonDatabinc	30	28	45
JacksonXml	3	1	3
Jsoup	34	35	39
JxPath	2	4	5
	273	254	357

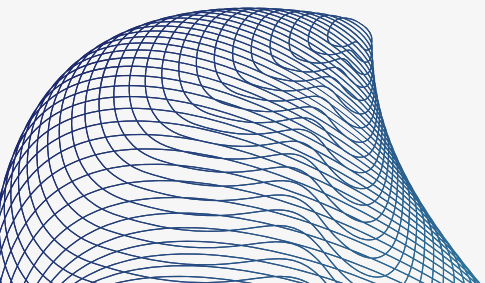
Model Accuracy

Model	Accuracy
GPT-3.5-Turb	52.2%
Coddegen	48.6%
Our Repair modul	71.2%

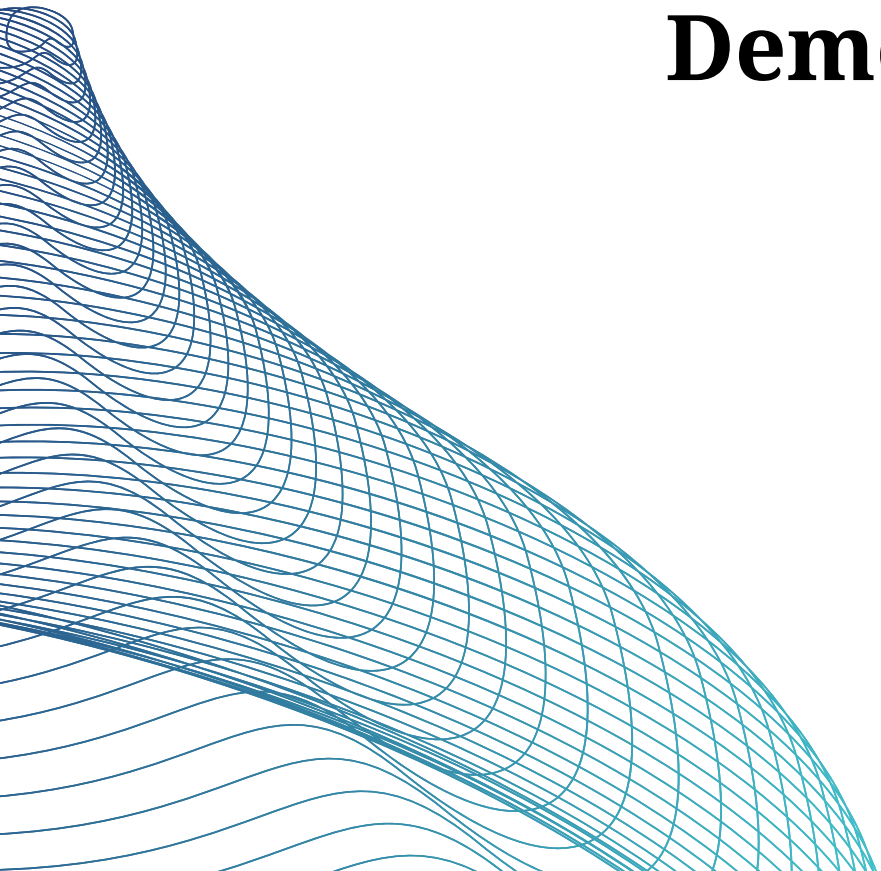
NOVELTY

- **Advanced Few-Shot Learning**: Integrated advanced few-shot learning mechanisms to boost repair effectiveness.

- **Auxiliary Information Integration**: Utilized repair-relevant information, such as bug reports and trigger tests, to improve repair accuracy and practicality.



Demonstration



Module 03

Analysis the Code Quality

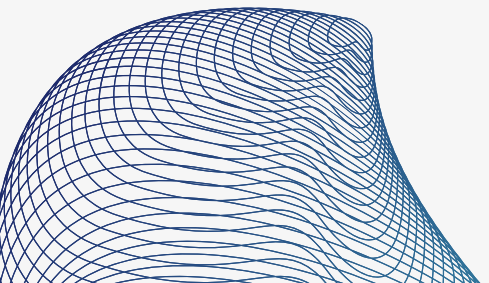
This module aims to analyze the quality of Java code snippets by leveraging machine learning techniques and a static code analysis, to predict various quality aspects of the code such as readability, maintainability, testability, efficiency. By analyzing code snippets, the module provides detailed feedback based on established code quality matrices, offering practical suggestions for code refactoring and improvement.

This feedback is tailored to help students understand specific areas where their code may have quality issues and how to improve code quality. The module focusses on ensuring that students learn to write cleaner, more professional, and efficient code.

RESEARCH GAP

- **Granular Feedback:** Existing systems lack specific, actionable feedback. Our project provides detailed, context-specific advice to guide students in addressing precise code issues.

- **Real-time Feedback:** Current solutions offer feedback post-submission, missing the chance to correct issues during coding. Our project integrates real-time feedback, giving immediate, adaptive advice.



SOLUTION IN DETAIL

Input – error free student's java code

Process

1. Data Gathering and preprocessing
2. Training the Model
3. Making Predictions and Evaluating the Model
4. Static Analysis with PMD
5. Feedback Generation

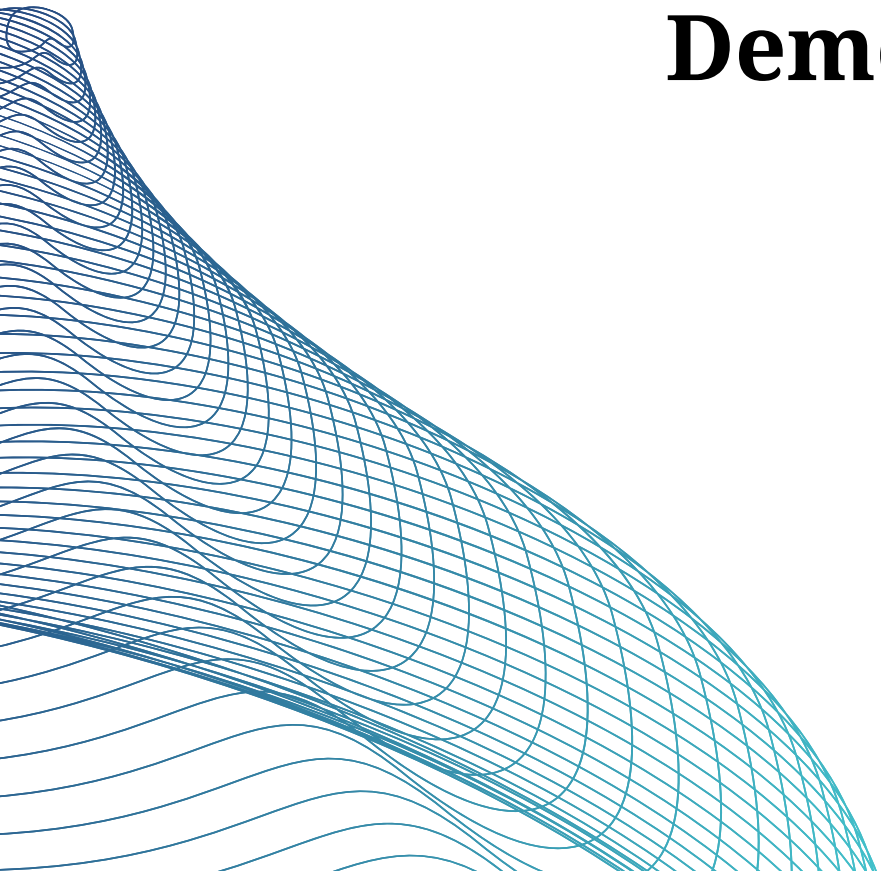
Output - Code quality feedback

Novelty

- Machine Learning-Based Code Analysis: Investigate the use of machine learning techniques for code quality analysis. Train models on a diverse set of Java code to identify patterns and provide feedback based on learned patterns.

- Personalize feedback based on individual student performance.

Demonstration



Code Vulnerability

- Weakness or flaw in a computer program's code that can be exploited by attackers to compromise the security of a system.
- Vulnerabilities often manifest themselves in subtle ways that are not obvious to code reviewers or the developers themselves.
- Vulnerabilities can lead to various security risks, such as unauthorized access, data breaches, denial of service, and more

Vulnerability issues

- **SQL Injection**
- **Cross-site scripting or XSS**
- **HTTP splitting attacks**
- **Command injection**
- **cryptographic issues**
- **Broken authentication and session management**
- **Cross site request forgery (csrf)**
- **Security Misconfigurations**

Novelty & Datasets

Novelty

vulnerability reports with educational resources. Provide links to documentation or explanations that help participants understand the nature of the vulnerabilities and how to fix them

Novelty Idea

Vulnerability: SQL Injection, Command injection..... more

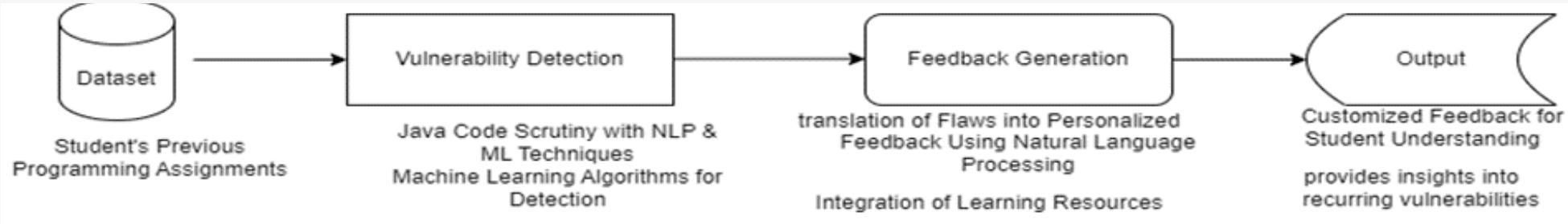
Description:
A potential SQL injection vulnerability was identified in the code. SQL injection occurs when user inputs are not properly validated, allowing attackers to manipulate SQL queries and potentially gain unauthorized access to the database.

Educational Resources:
1. [OWASP SQL Injection Prevention Cheat Sheet] (https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
2. [Web Security Academy: SQL Injection](https://portswigger.net/web-security/sql-injection)
3. [SQL Injection: Understanding and Prevention](https://www.acunetix.com/blog/web-security-zone/sql-injection-understanding-prevention/)

- [Video Tutorial: Fixing SQL Injection Vulnerabilities in Java](#)

Recommended Fix and Refactored code :
1. Utilize parameterized queries or prepared statements to separate user input from SQL queries.

Vulnerability Management Process



Evaluation with Existing Research

Existing Research :

- High false positives in static analysis.
- Resource-intensive dynamic analysis.
- Missing untriggered vulnerabilities.

My Approach:

- Better Accuracy: Fewer false positives with CodeBERT.
- Resource Efficient: Less computational demand.
- Detailed Feedback: LLaMA 3 provides explanations and fixes.

VII. Experimental Results

Highlights of our evaluation results are:

- FUNDED delivers, on average, a 92% accuracy, for software vulnerability detection (Sec. VII-A and Sec. VII-B).
- FUNDED outperforms all competing methods for detecting (Sec. VII-C) and collecting (Sec. VII-E vulnerable code).
- We provide detailed analysis for the working mechanisms of FUNDED (Sec. VII-G and Sec. VII-F).

Year of Published -2022

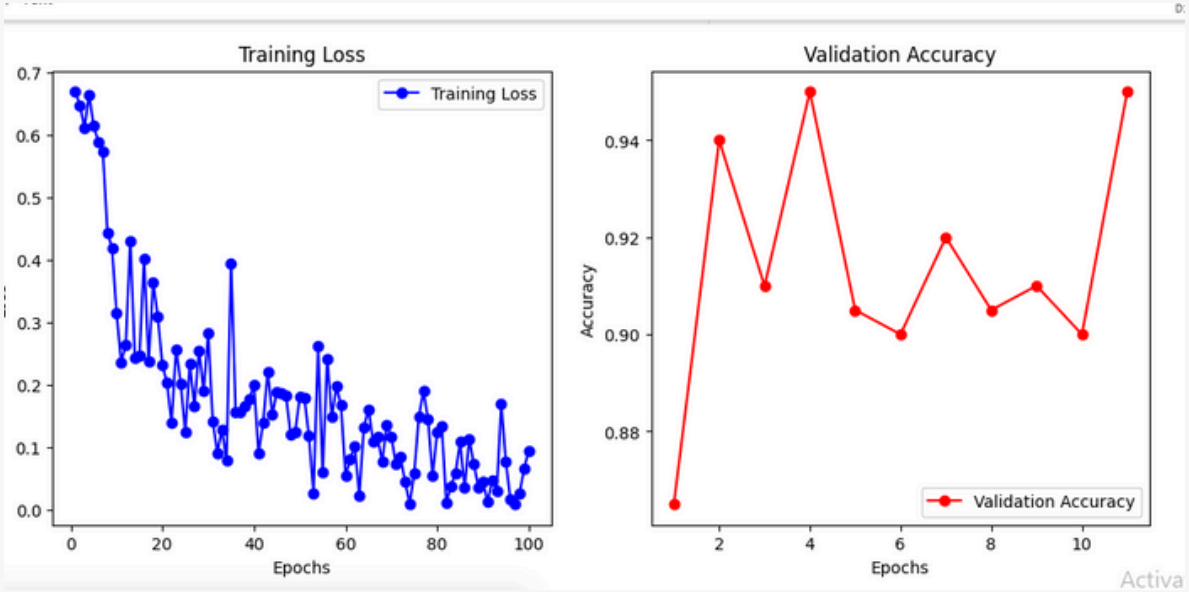
My Model Result:

Epoch	Training Loss	Validation Loss	Accuracy
1	0.613800	0.566257	0.690000
2	0.293700	0.225987	0.925000
3	0.360100	0.208760	0.900000
4	0.215200	0.154884	0.945000
5	0.132400	0.177588	0.950000
21	0.017400	0.473046	0.935000
22	0.026700	0.470702	0.935000
23	0.024700	0.507746	0.930000
24	0.039300	0.533045	0.930000
25	0.009800	0.528068	0.930000

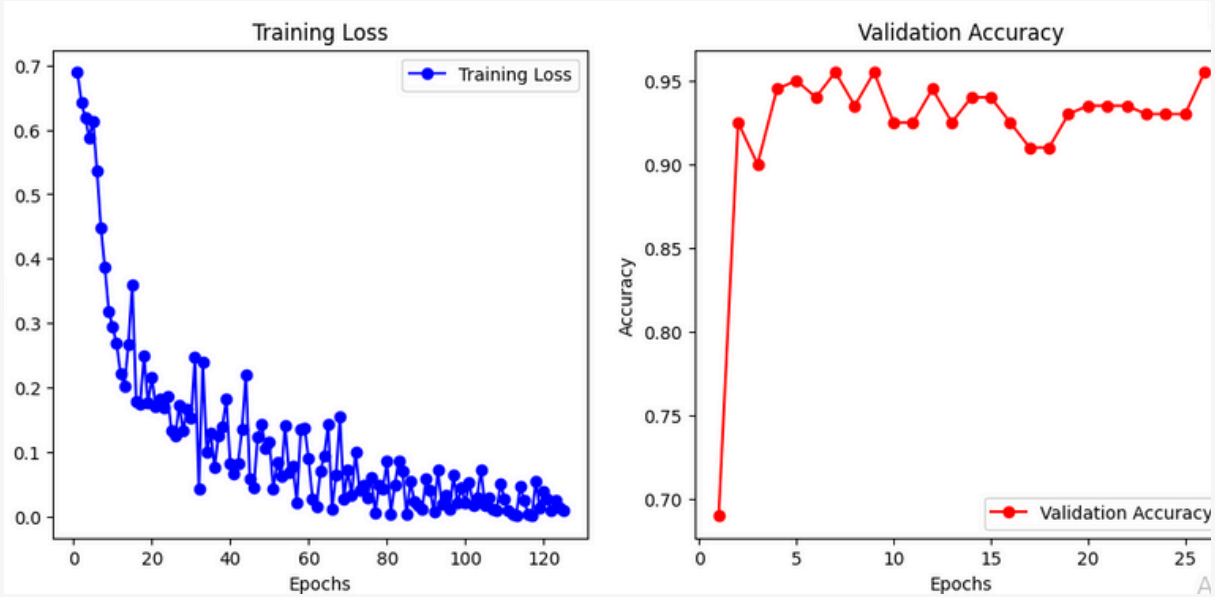
[25/25 00:02]

Validation Accuracy: 0.955

Evaluation

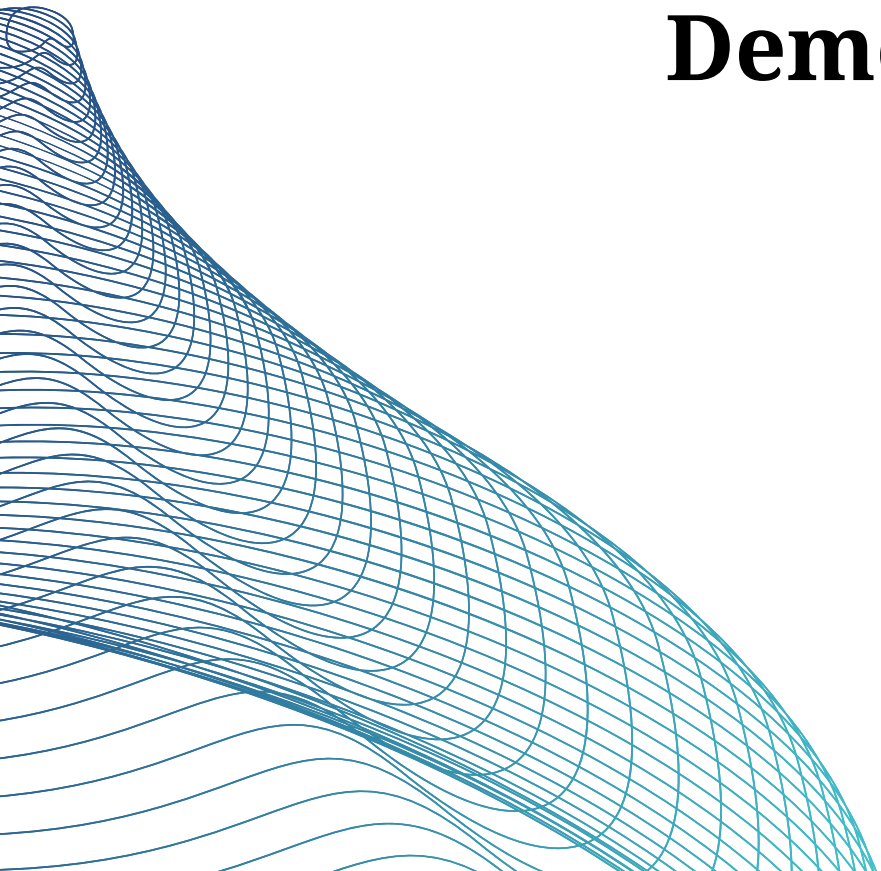


train_batch_size=8
train_epochs=10



train_batch_size=16
train_epochs=25

Demonstration



References

- [1] H. Keuning, B. Heeren, and J. Jeuring, “Strategy-based feedback in a programming tutor,” in Proceedings of the Computer Science Education Research Conference, Berlin Germany: ACM, Nov. 2014, pp. 43–54. doi: 10.1145/2691352.2691356.

- [2] A. Estey, H. Keuning, and Y. Coady, “Automatically Classifying Students in Need of Support by Detecting Changes in Programming Behaviour,” in Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, Seattle Washington USA: ACM, Mar. 2017, pp. 189–194. doi: 10.1145/3017680.3017790.

- [3] A. Birillo et al., “Detecting Code Quality Issues in Pre-written Templates of Programming Tasks in Online Courses,” in Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, Turku Finland: ACM, Jun. 2023, pp. 152–158. doi: 10.1145/3587102.3588800.

- [4]H. Keuning, J. Jeuring, and B. Heeren, “Towards a Systematic Review of Automated Feedback Generation for Programming Exercises,” in Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, Arequipa Peru: ACM, Jul. 2016, pp. 41–46. doi: 10.1145/2899415.2899422.

Thank You .....