

Akash Yadav

SOFTWARE_DEFECT_PREDICTION_FOR_PYTHON_SOFTWARE...

 Mtech Thesis

Document Details

Submission ID

trn:oid:::27535:139851647

Submission Date

May 21, 2026, 1:40 PM GMT+5:30

Download Date

May 21, 2026, 1:59 PM GMT+5:30

File Name

SOFTWARE_DEFECT_PREDICTION_FOR_PYTHON_SOFTWARE_USING_ENSEMBLE_LEARNING__EXPER....pdf

File Size

503.0 KB

42 Pages

8,693 Words

49,213 Characters

09% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text

Match Groups

-  **76 Not Cited or Quoted 12%**
Matches with neither in-text citation nor quotation marks
-  **0 Missing Quotations 0%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 8%  Internet sources
- 4%  Publications
- 10%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- 76 Not Cited or Quoted 12%**
Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%**
Matches that are still very similar to source material
- 0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 8% Internet sources
- 4% Publications
- 10% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	dspace.dtu.ac.in:8080	3%
2	Internet	www.dspace.dtu.ac.in:8080	1%
3	Student papers	Delhi Technological University on 2026-05-21	<1%
4	Student papers	Delhi Technological University on 2026-05-21	<1%
5	Internet	andaloussi.org	<1%
6	Internet	www.um.edu.mt	<1%
7	Student papers	Suleyman Demirel University, Kazakhstan on 2024-05-29	<1%
8	Student papers	Indian Institute of Technology, Madras on 2018-06-11	<1%
9	Publication	"AI Technologies for Information Systems and Management Science", Springer Sc...	<1%
10	Publication	Yunqiu Sun, Zhiyu Sun, Zhiman Jiang. "The environmental Kuznets curve hypothe...	<1%

11	Internet	arxiv.org	<1%
12	Internet	dl.lib.uom.lk	<1%
13	Internet	es.scribd.com	<1%
14	Internet	www.tnsroindia.org.in	<1%
15	Student papers	Hong Kong Baptist University on 2014-04-28	<1%
16	Student papers	Queen's University of Belfast on 2023-09-11	<1%
17	Publication	Ruchika Malhotra, Priya Singh. "A Unified Deep Learning Based Feature Represen...	<1%
18	Student papers	The Scientific & Technological Research Council of Turkey (TUBITAK) on 2022-05-27	<1%
19	Student papers	University of Southampton on 2019-05-01	<1%
20	Student papers	University of Strathclyde on 2026-04-01	<1%
21	Internet	real-j.mtak.hu	<1%
22	Internet	www.mdpi.com	<1%
23	Student papers	Midlands State University on 2025-10-30	<1%
24	Student papers	University of East London on 2025-09-08	<1%

25	Student papers	Dublin Business School on 2025-08-28	<1%
26	Publication	H L Gururaj, Francesco Flammini, V Ravi Kumar, N S Prema. "Recent Trends in He...	<1%
27	Publication	Harvinder Singh, Priyanka Kaushal, Sarabjeet Kaur. "Advanced Computing and AI...	<1%
28	Student papers	Khalifa University of Science Technology and Research on 2018-07-29	<1%
29	Publication	Ruchika Malhotra, Priya Singh. "DHG-BiGRU: Dual-attention based hierarchical ga...	<1%
30	Student papers	University of East London on 2026-05-06	<1%
31	Internet	academic.oup.com	<1%
32	Student papers	iGroup on 2013-09-11	<1%
33	Student papers	iGroup on 2014-01-26	<1%
34	Internet	moam.info	<1%
35	Internet	vdoc.pub	<1%
36	Internet	xin-xia.github.io	<1%
37	Publication	Alfalasi, Rashed. "Deepfake Audio Detection.", Rochester Institute of Technology	<1%
38	Student papers	College Users IQAC, Library, Others on 2026-01-05	<1%

39	Publication	Haining Meng, Han Wu, Xiaoqing Li, Xinhong Hei. "GLA-SDP: A novel attention-ba...	<1%
40	Student papers	Higher Education Commission Pakistan on 2026-01-27	<1%
41	Student papers	Liverpool John Moores University on 2020-08-09	<1%
42	Student papers	Liverpool John Moores University on 2022-07-30	<1%
43	Student papers	Multimedia University on 2026-04-01	<1%
44	Student papers	National College of Ireland on 2024-12-11	<1%
45	Publication	Nouri Gooshki, Sadegh. "A Scalable Deep Learning Framework for Autonomous R...	<1%
46	Student papers	The Stockholm School of Economics in Riga on 2011-04-11	<1%
47	Student papers	The University of Manchester on 2015-02-05	<1%
48	Student papers	Universiti Kebangsaan Malaysia on 2024-05-03	<1%
49	Student papers	University of Birmingham on 2025-03-28	<1%
50	Student papers	University of Lincoln on 2025-08-03	<1%
51	Internet	dokumen.pub	<1%
52	Internet	encyclopedia.pub	<1%

53	Internet	scpe.org	<1%
54	Internet	tudr.thapar.edu	<1%
55	Publication	Alfredo Daza. "Software Defect Prediction Based on a Multiclassifier with Hyperp...	<1%
56	Student papers	Edith Cowan University on 2022-12-12	<1%
57	Publication	Jingxiu Yao, Martin Shepperd. "The impact of using biased performance metrics o...	<1%
58	Publication	Kimball, Richard. "SOCDASH: A Scalable Occupant Centric Data Predictive Control ...	<1%
59	Student papers	University of Science and Technology of Fujairah on 2026-05-10	<1%
60	Student papers	University of Sydney on 2017-06-09	<1%

SOFTWARE DEFECT PREDICTION FOR PYTHON SOFTWARE USING ENSEMBLE LEARNING: EXPERIMENTAL ANALYSIS AND REVIEW OF SEQUENTIAL PREDICTION MODELS

A DISSERTATION

Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

submitted by

AKASH YADAV

(24/RSE/05)

Under the Supervision of

Mrs. Priya Singh

Assistant Professor, Department of Software Engineering

Delhi Technological University



Department of Software Engineering

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daultapur, Main Bawana Road, Delhi-110042, India

May, 2026

DEPARTMENT OF MECHANICAL ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, AKASH YADAV Roll No – 24/RSE/05 students of M.Tech BY Research (SOFTWARE ENGINEERING), hereby declare that the Dissertation titled **SOFTWARE DEFECT PREDICTION FOR PYTHON SOFTWARE USING ENSEMBLE LEARNING: EXPERIMENTAL ANALYSIS AND REVIEW OF SEQUENTIAL PREDICTION MODELS** which is submitted by us to the SOFTWARE ENGINEERING, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

AKASH YADAV

Date: 20.05.25

DEPARTMENT OF MECHANICAL ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled **SOFTWARE DEFECT PREDICTION FOR PYTHON SOFTWARE USING ENSEMBLE LEARNING: EXPERIMENTAL ANALYSIS AND REVIEW OF SEQUENTIAL PREDICTION MODELS** which is submitted by AKASH YADAV, Roll No – 24/RSE/05, SOFTWARE ENGINEERING, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Mrs. PRIYA SINGH

Date: 20.05.2026

SUPERVISOR

DEPARTMENT OF MECHANICAL ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

I wish to express our sincerest gratitude to Mrs. PRIYA SINGH for the continuous guidance and mentorship that she provided us during the project. She showed us the path to achieve our targets by explaining all the tasks to be done and explained to us the importance of this project as well as its industrial relevance. She was always ready to help us and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Delhi

AKASH YADAV

Date: 20.05.2026

Abstract

Software Defect Prediction is another sub field of Software Quality Assurance that is a core area and aims to identify which parts are more likely to be faulty during early stages of the development life cycle, allowing the teams to better allocate testing efforts on areas that are more likely to fail. Most of the previous work has been limited to C, C++, and Java programming languages using the standard NASA and PROMISE datasets, although some gaps exist. On the one hand, there is still little guidance for Python specific defect prediction, while on the other, there does not seem to be a clear guidance in the literature regarding which sequential deep learning architectures perform better. The two peer reviewed contributions contained within this thesis examine both of these concerns. This thesis aims to fill both of these gaps in two peer reviewed contributions. The first paper presents an empirical evaluation of five ensemble methods – XGBoost, LightGBM, CatBoost, Stacking Classifier, and Voting Classifier on a dataset derived from PyTraceBugs dataset of 638,568 class instances with 18 static code metrics and a binary defect label. Random Oversampling is employed in order to reduce the high degree of class imbalance (buggy instances are 0.038% of the total). The features extraction is done in two stages, resulting in a set of 15 features. The Stacking Classifier achieved the highest F1-score, 0.6966 and MCC, 0.6986 while the Voting Classifier achieved the highest Precision, 0.7632 and LightGBM achieved highest AUC-ROC 0.9455, demonstrating good discriminative capacity under extreme imbalance. The second contribution is a systematic study of 36 peer-reviewed papers, published between 2018 and 2025, that investigate the use of RNNs, LSTMs, GRUs, BiLSTMs and hybrid attention-based models for SDP, in which five research questions are answered about existing model taxonomies, datasets, evaluation methods, state of the art performance, and challenges. Among others, some key findings include the use of hybrid and LSTM- based models in the literature, the use of PROMISE repository as a benchmark, and the dominance of Java language pointing

to the need of research on Python. The most efficient model is the CodeBERT-BiGRU model with an accuracy of 93%, F1 score of 88.9% and MCC of 0.85.



1

Contents

Candidate's Declaration	i
Certificate	ii
Acknowledgement	iii
Abstract	v
List of Tables	viii
List of Figures	ix
List of Symbols, Abbreviations	x
Abbreviations	x
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Problem Statement	1
1.3 Research Objectives	2
2 LITERATURE REVIEW	3
2.1 Overview	3
2.2 Traditional Machine Learning Approaches	3
2.3 Ensemble Learning Methods	4
2.4 Sequential Deep Learning Models	4
2.5 Review Methodology	5
2.6 RQ1: Seq-SDP Model Taxonomy	5
2.7 RQ2: Datasets in Seq-SDP Research	8
2.8 RQ3: Evaluation Metrics in Seq-SDP	9
2.9 RQ4: State-of-the-Art Seq-SDP Performance	10
2.10 RQ5: Challenges and Limitations of Seq-SDP Models	10
2.11 Summarised Findings	12
3 Methodology	13

16

3.1	Overview	13
3.2	Dataset Description	13
3.3	Dataset Preprocessing	14
3.4	Feature Selection	14
3.5	Train/Test Split and Resampling	16
3.6	Feature Scaling	16
3.7	Ensemble Classifiers	17
3.7.1	XGBoost	17
3.7.2	LightGBM	17
3.7.3	CatBoost	17
3.7.4	Stacking Classifier	17
3.7.5	Voting Classifier	18
3.8	Hyperparameter Configuration	18
3.9	Performance Metrics	19

4 Results and Discussion 20

4.1	Overview	20
4.2	Classification Performance	20
4.2.1	Stacking Classifier	21
4.2.2	Voting Classifier	21
4.2.3	LightGBM	21
4.2.4	XGBoost and CatBoost	21
4.3	Time and Memory Efficiency	21
4.4	Visual Evaluation	22
4.4.1	ROC Curve Analysis	22
4.4.2	Confusion Matrix Analysis	22
4.5	Comparative Analysis and Model Selection	24
4.6	Synthesis with Sequential Model Findings	24

5 CONCLUSION AND FUTURE SCOPE 25

5.1	Implications of the Implementation Study	25
5.2	Implications of the Sequential Model Review	26
5.3	Limitations	26
5.4	Future Work	26

A LIST OF PUBLICATION 27

List of Tables

2.1	Comparison of Seq-SDP Models	8
2.2	Comparison of evaluation metrics for Seq-SDP	11
3.1	Description of Metrics in the PyTraceBugs Dataset	15
3.2	Tuned Hyperparameters for All Models	18
4.1	Performance Metrics of Ensemble Models	20
4.2	Computational Efficiency of Ensemble Models: Training Time, Testing Time, and Memory Usage	22

List of Figures

12	2.1 Year-wise distribution of research papers published (2018–2025). 6
	2.2 Distribution of publishers for reviewed research papers. 6
	2.3 Distribution of Seq-SDP models 7
	2.4 Number of all the Datasets used in different studies 9
	3.1 Proposed ensemble-based defect prediction framework for Python code. . . 13
	3.2 op 15 selected features 16
44	4.1 ROC Curves of Ensemble Models 23
	4.2 Confusion Matrices of Ensemble Models 23

List of Symbols

	r	Radius, m
	α	Angle of thesis in degrees
	β	Flight path in degrees

Chapter 1

INTRODUCTION



1.1 Motivation

Hidden bugs are becoming more expensive with the advent of advanced software systems in healthcare, finance, and transportation. Testing cannot be thorough because of resource constraints for testing and also due to the unequal distribution of defects. This problem has been solved using Software Defect Prediction (SDP), which analyzes data from past experience along with other software metrics to identify potential risky modules in advance. The problem of SDP is classified as a binary classification problem and has gained many supervised machine learning techniques [1]. Three generations of SDP studies have been developed: traditional statistical and machine learning algorithms, which were hindered by the fact that they were unable to detect non-linear relationships and thus had limitations in predictive performance [2]; the ensemble learning algorithms Random Forest, XGBoost, LightGBM, and CatBoost, which had better performance [3, 4]; and finally deep sequential networks, like RNN, LSTM, GRU, and BiLSTM, which could model the patterns of defects in commits [5, 6]. Nevertheless, there are still some shortcomings of SDP research. Another is that almost all of the researches have been conducted with C, C++ and Java programming languages, despite the fact that Python gained popularity in machine learning tasks owing to its own unique features. Secondly, no review paper has covered the sequential SDP paradigm, datasets, evaluation approaches and challenges yet. In this respect, the current work aims at filling both gaps.

1.2 Problem Statement

The main issues discussed in this thesis include lack of empirical study on SDP on Python programs, considering the increase in usage of Python in current software development practices; class imbalance, which is highly prevalent in real datasets of defects with defective samples being only 0.038% of the total samples; even though ensemble **learning techniques have been proven to be effective in** predicting defects using Java and C pro-

grams, no systematic comparison has been done on modern gradient boosting algorithms XGBoost, LightGBM, and CatBoost and meta-ensemble techniques such as Stacking and Voting under imbalance. Finally, with the fast increase in the number of studies in Seq-SDP, there seems to be confusion regarding what architectures should be considered, datasets that are used, evaluation metrics, and problems that still need to be addressed in order to find an exact research direction.

1.3 Research Objectives

The main objectives that will be accomplished by the research work are as follows:

1. Design and validate an experiment to create a software defect prediction dataset from static code metrics obtained from the PyTraceBugs repository using Python programming language.
2. Design and implement an entire software defect prediction system involving data pre-processing, feature selection, class imbalance problem resolution, feature scaling and optimization of hyperparameters.
3. Carry out a comparative evaluation of ensemble learning models such as XGBoost, LightGBM, CatBoost, Stacking Classifier and Voting Classifier using several performance and efficiency measures.
4. Carry out a systematic review of latest research works done on sequential seq-SDP to evaluate model architecture, data sets, performance measures and open issues.
5. Finally, draw an informed conclusion based on findings from experiments and review.

Chapter 2

LITERATURE REVIEW

2.1 Overview

SDP has been a thriving research area for more than three decades, advancing from rudimentary statistical models to sophisticated deep learning models. In this chapter, we discuss the related literature in three eras: machine learning era, ensemble learning era, and sequential deep learning era. We further discuss the future trend in SDP including embedding and attention models.

2.2 Traditional Machine Learning Approaches

The first wave of SDP included the application of traditional machine learning techniques to metrics generated from source code. Metrics like number of lines of code, cyclomatic complexity, code changes, and object coupling were fed into algorithms like NB (Naive Bayes), DT (Decision Trees), LR (Logistic Regression), and SVM (Support Vector Machine) to classify modules as having defects or not having defects.

One such early study carried out by Arisholm et al. [2] involved a comparative analysis of the applicability of various data mining techniques in forecasting fault-proneness in telecommunications software projects. These types of studies served as important benchmarks that would be used in the assessment of future models developed in the domain. On the other hand, some of the weaknesses associated with early SDP models were mainly due to the inadequacy in obtaining precise and well-defined labeled data relevant to each software project.

The use of feature selection and undersampling techniques with the AdaBoost technique was investigated by Khoshgoftaar et al. [7], and it was found that the boosting technique helps to boost the classification outcomes considerably. The usefulness of software dependencies and software churn for predicting field failures was discovered by Nagappan and Ball [8], and hence the process level metrics emerged as a natural consequence of the traditional code metrics used for the product level analysis. Huda et al. [9] have proposed

an approach to SDP based on filter/wrapper feature selection and ensemble classification.

2.3 Ensemble Learning Methods

The main drawback of individual classifiers is overcome by ensemble learning approaches in which multiple base classifiers are combined to improve the predictions made by any individual base classifier. The two major types of ensembles are sequential and parallel [10]. The sequential type of ensemble learning techniques involves the use of boosting algorithms that include AdaBoost, Gradient Boosting, XGBoost, LightGBM, and CatBoost. In sequential ensembles, multiple base classifiers are trained in sequence with each classifier learning from the mistakes of the previous classifiers [11].

The work by Laradji et al. [3] proved that applying ensemble learning to well-chosen features leads to better SDP results than using single models, thereby laying the groundwork for the paradigm of ensemble learning after feature selection. The research conducted by Shanthini and Chandrasekaran [12] on the bagging ensemble method using SVM as the base classifier revealed that the ensemble version of SVM performs better than the standalone version when predicting software faults.

Recently, Dong et al. [4] conducted a systematic assessment of ensemble-based SDP techniques with homogeneous and heterogeneous types of ensembles using different benchmark data sets. In addition, Mafarja et al. [13] introduced a feature selection framework using the whale optimization algorithm along with Random Forest ensemble technique and reported promising outcomes for NASA data set for defect prediction. Arar and Ayan [14] presented that a feature-dependent NB classifier is capable of removing redundant features.

XGBoost [15] was a landmark development for gradient boosting due to its efficient use of L1 and L2 regularization, column sampling, and second order gradient calculations. LightGBM was an improvement to XGBoost in terms of computational efficiency by virtue of histogram-based gradients and leaf-wise splits. On the other hand, CatBoost offered a unique method called ordered boosting that resolved the problem of shifting predictions in categorical variables.

2.4 Sequential Deep Learning Models

Deep Learning models operating sequentially are the state-of-the-art models in SDP literature. Hochreiter and Schmidhuber [5] proposed the LSTM (Long Short Term Memory) network model that solves the vanishing gradient problem faced by conventional RNN models using a gating system. The LSTM network is a very efficient method for making predictions on defects at the commit level because the order of the commit is an important

factor [16]. Cho et al. [17] came up with the GRU (Gated Recurrent Unit) network model as a light-weight solution to LSTM networks.

One of the early frameworks for defect prediction using deep learning is DeepSoft [18]. This framework uses deep learning to show that learned representations outperform manually crafted features in defect prediction. On PROMISE datasets, Munir et al. [6] introduced a hybrid model using GRU-LSTM with attention mechanism. The integration of CNN and GRU with SMOTE-Tomek resampling has been introduced by Khleel and Nehéz [19] which showed good performance on highly imbalanced defect data sets. Krill herd with stacked LSTM and attention mechanism has also been used in [20].

The current state-of-the-art approach to Seq-SDP is known as the CodeBERT-BiGRU architecture [21], which combines contextualized embeddings of code generated using CodeBERT along with a BiGRU, providing an accuracy of 93% and F1-score of 88.9%. In the DHG-BiGRU approach [22], dual attention was used across time and space, adding the element

2.5 Review Methodology

The systematic review on studies that employed Seq-SDP (Sequential Software Defect Prediction) models during 2018 to 2025 has been conducted. In total, 36 scholarly articles were identified for review, including IEEE, Springer, Elsevier/ACM, MDPI, and other journals. **Year-wise distribution is shown in Figure 2.1**, which reflects a steady increasing pattern with a peak in 2022. There are five research questions (RQ1–RQ5) that have been reviewed altogether.

The publisher distribution chart of the studies analyzed can be seen from Figure 2.2. It is apparent that IEEE leads the publication numbers with 40.5%, with Springer coming in second at 21.6% and Elsevier/ACM at third place with 8.1%. The other publications include MDPI, Hindawi, PLOS ONE, and PeerJ, among others.

2.6 RQ1: Seq-SDP Model Taxonomy

RQ1: What are the various Seq-SDP models that have been applied in software defect prediction?

Seq-SDP is based on a wide variety of neural architectures suited for sequential/time-series data processing. While regular RNN is the basic architecture, its major disadvantage lies in the issue of vanishing gradients, rendering it incapable of modeling long-distance dependencies in software artifacts sequences [23]. The problem was addressed by using LSTM with its input, output, and forget gates that allow preserving meaningful information irrespective of the sequence length [5]. Another approach for the same is

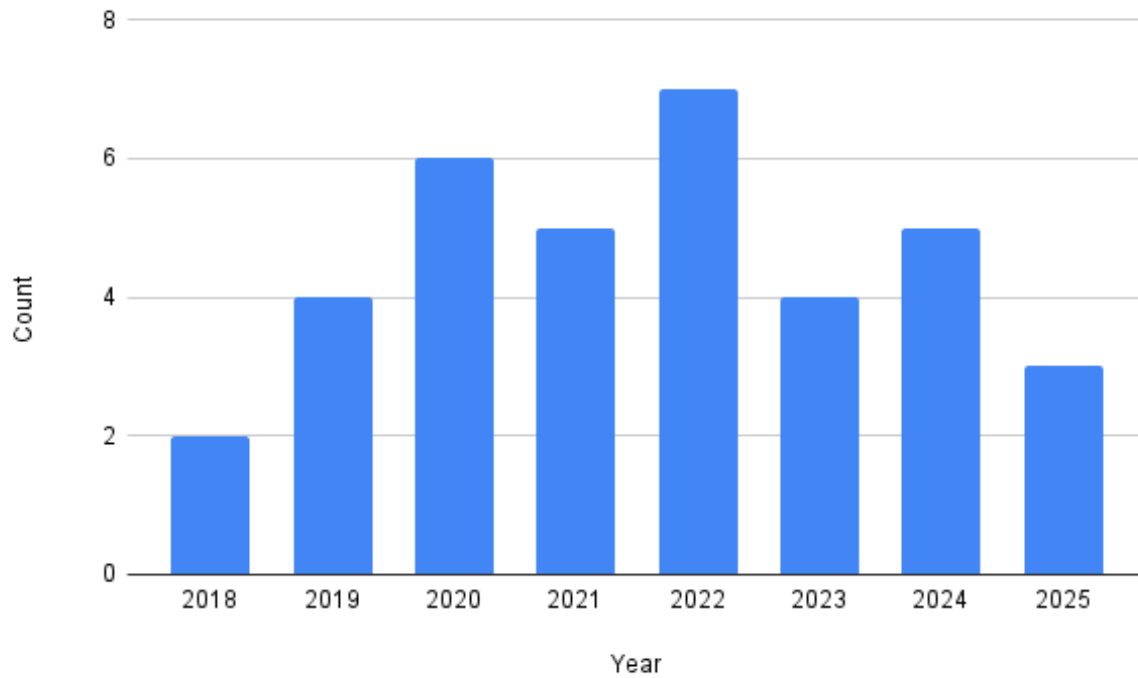


Figure 2.1: Year-wise distribution of research papers published (2018–2025).

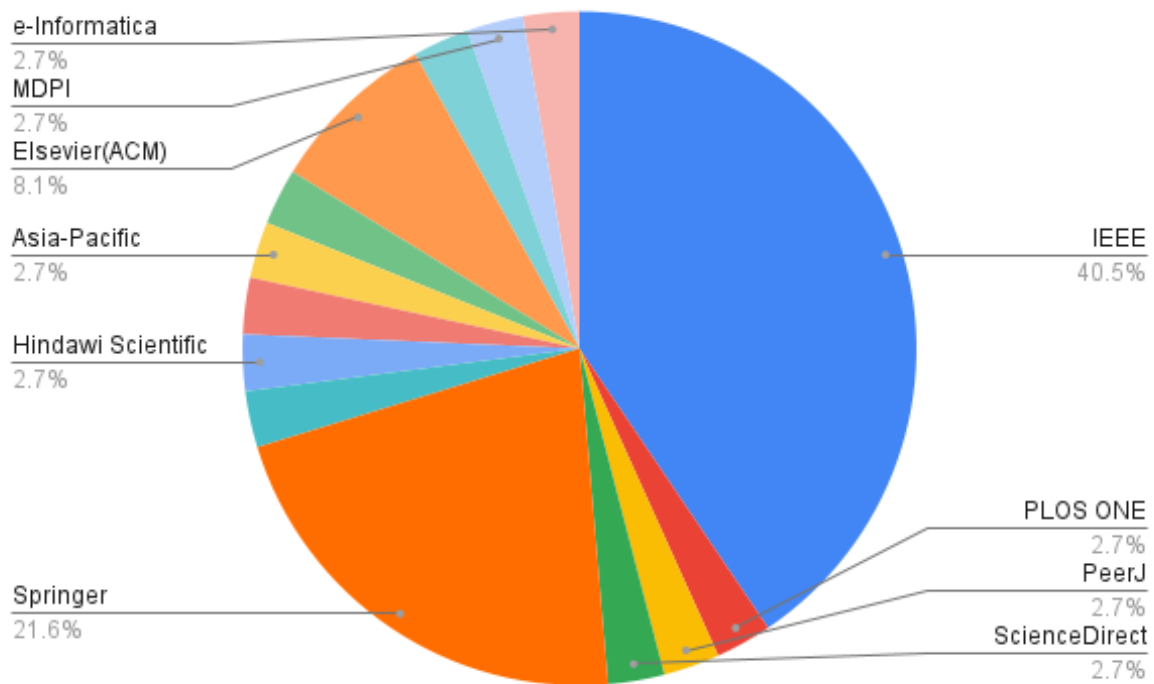


Figure 2.2: Distribution of publishers for reviewed research papers.

GRU, offering reduced computational costs compared to LSTMs while still maintaining similar capabilities [17].

Reasons why one would be interested in investigating the different kinds of Seq-SDP

models include Recurrent Neural Networks, LSTM, and GRU are due to their revolutionary nature when applied in the domain of software defect prediction. It becomes important for researchers to explore the scope and applications of these models to understand emerging trends, overcome existing shortcomings, and look for avenues that can enhance the state-of-the-art in the research community. The Table ?? below presents an overview of different Seq-SDP models, which include RNN, LSTM, GRU, and Attention Mechanisms, highlighting their definitions, strengths, and shortcomings in the context of software defect prediction. The Table ??, highlights the analysis of different Seq-SDP models utilized for predicting software defects prediction. From Fig. 2.3, it is clear

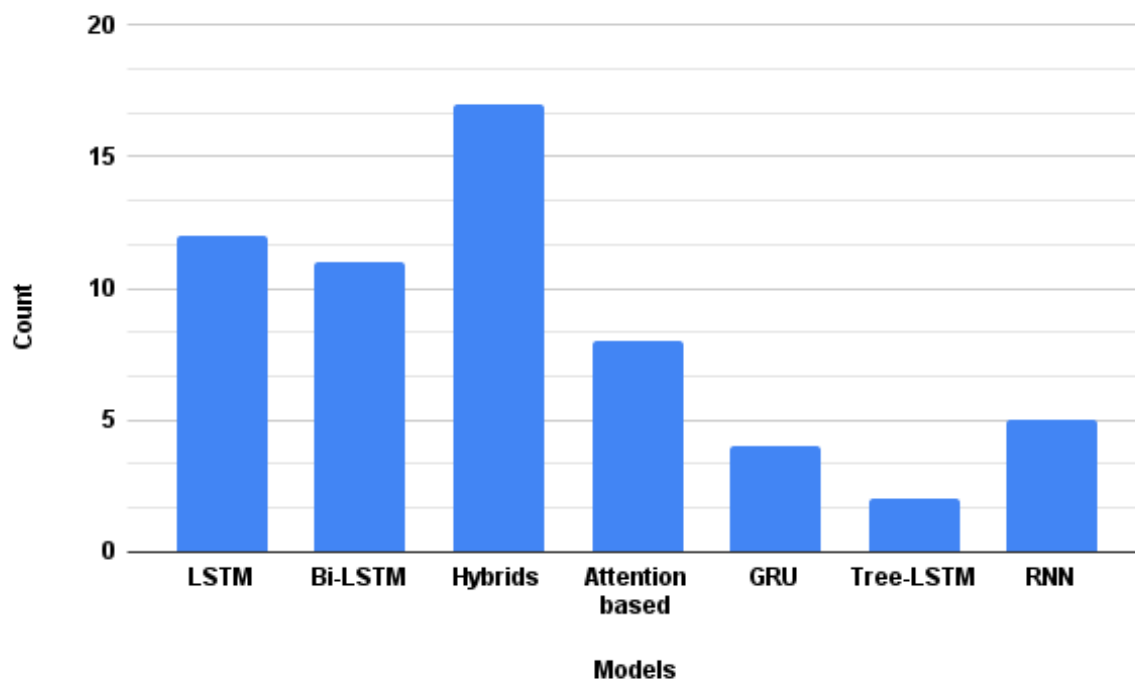


Figure 2.3: Distribution of Seq-SDP models

that the models using both techniques together (hybrid technique) like combining several Seq-SDP models and LSTM models have become popular among defect prediction studies. Table 2.1 demonstrates the comparative perspective of the major groups of Seq-SDP models.

As shown in Fig. 2.3, the distributions of the model architectures used in the 36 papers are displayed. Among all the models, hybrid models and LSTM networks have higher usage frequency than other kinds of models, occupying more than 60% of the papers under discussion.

Table 2.1: Comparison of Seq-SDP Models

Model	Definition	Advantages	Drawbacks
LSTM	Advanced RNN with gates for long-term memory.	Learns long sequences effectively, overcomes vanishing gradient issue.	Computationally intensive with longer training time.
GRU	Simplified LSTM with fewer gates.	Faster training, memory efficient, less complexity.	May capture less long-term dependency than LSTM.
BiLSTM	Bidirectional LSTM reading input in both directions.	Captures past and future context simultaneously.	Higher computational cost than unidirectional models.
Attention + RNN	RNN enhanced with attention to focus on relevant sequence parts.	Improves feature importance extraction and performance.	Requires more computation and training resources.
Attention + LSTM	LSTM with attention layer to boost feature selection.	Handles complex sequences better, improves accuracy.	Computationally expensive, slower training.

2.7 RQ2: Datasets in Seq-SDP Research

RQ2: What are the various datasets commonly employed in the Seq-SDP domain?

The empirical evaluation of Seq-SDP models depends critically on the quality and diversity of benchmark datasets. The most significant datasets in the literature include:

- **NASA MDP Datasets:** The NASA Metrics Data Program provides **static code metrics** with **defect labels from real-world NASA projects**. Key datasets include CM1, KC1, and JM1 [24]. Most of the baseline comparison are done on these datasets.
- **PROMISE Repository:** Popular benchmark dataset used in SDP research [24]. Provides benchmarks such as Apache projects and MC1 to be used for defect prediction experiments.
- **Eclipse IDE Datasets:** Data collected from Eclipse IDE based projects like JDT and PDE. They offer detailed information regarded defects and version control and, hence are very useful for practical purposes [25].
- **Code4Bench:** Structured baseline dataset created specifically to test machine learning models on software engineering tasks such as SDP tasks. [26].
- **Datasets Based on GitHub Sources:** Datasets built upon repositories from

GitHub with open source code written in Python, Java, and C++. All of these needs thorough preprocessing prior to analysis [27].

- **PyTraceBugs:** Dataset used in implementation experiment of this thesis for SDP using ensemble models [28]. It includes source files in Python with static code metrics collected using SciTools Understand.

The distribution of data sets used in each paper among the total 36 is in Figure 2.4. PROMISE repository stands out significantly in using for SDP due to its years of the industry standard benchmark for SDP task. The deficiency of Python-based data sets such as PyTraceBugs shows the problem addressed by this thesis.

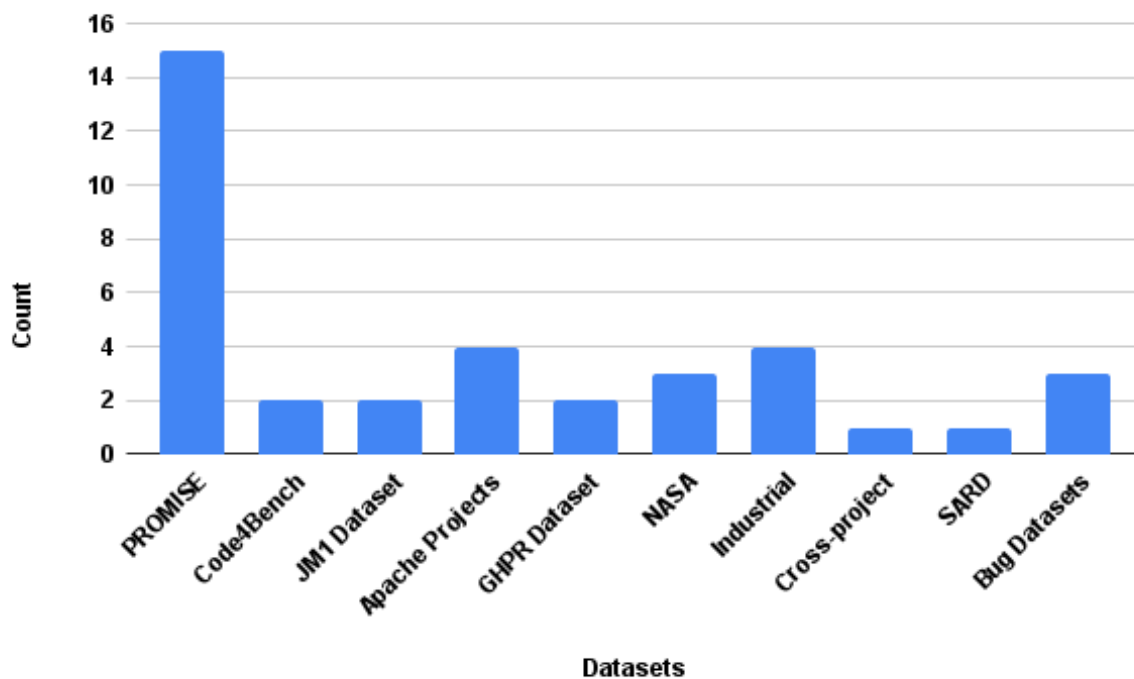


Figure 2.4: Number of all the Datasets used in different studies .

Analysis of programming languages based on the selected publications proves that Java is the most popular language, C/C++ being the second most popular one, while use of Python in studies makes only a minor part of this body of research, which proves the potential of this thesis topic.

2.8 RQ3: Evaluation Metrics in Seq-SDP

RQ3: How do different evaluation metrics perform in Seq-SDP models?

Evaluating SDP models requires careful selection of metrics that accurately reflect performance under the class imbalance characteristic of most defect datasets. Ta-

ble 2.2 provides a comprehensive summary of the evaluation metrics employed in the reviewed Seq-SDP papers, including the number of papers using each metric, its definition, and its relative strengths.

The most common metric used is the F1-score, highlighting the fact that the community has realized that accuracy is not sufficient when dealing with class imbalance. The MCC is becoming increasingly popular due to its effectiveness in dealing with the four aspects of the confusion matrix, according to Yao and Shepperd [29].

2.9 RQ4: State-of-the-Art Seq-SDP Performance

RQ4: Which Seq-SDP models outperform existing approaches?

Of all the models surveyed, the model called CodeBERT-BiGRU [21] stands out as the most advanced currently available, demonstrating an accuracy rate of 93%, 88.7% precision, 89.1% recall, 88.9% F1-score, and 0.85 MCC. Studies have proven that both elements combined yield better results than either one used separately.

DHG-BiGRU model [22] uses dual attention over both temporal and spatial dimensions and has better performance in terms of accuracy, precision, recall, and MCC scores than baseline models. The visualized attention weight of the model helps provide interpretability about defect causes.

The architectures that integrate more than one Seq-SDP models (e.g., GRU-LSTM architecture, BiLSTM with attention, and transformers combined with a recurrent decoder) outperform single-architecture systems consistently. The integration of pre-trained embeddings (e.g., Word2Vec, FastText, and CodeBERT) also enhances performance as the initial representation of the code tokens and commit messages is enriched

Transfer learning and fine-tuning approaches have proved to be very effective in cross-project settings, achieving excellent results in terms of precision, recall, and F1 [30, 31]. Attention-based models have exhibited enhanced ability to extract important features, and attention weights can give an indication of which parts of code contribute most to the presence of bugs

2.10 RQ5: Challenges and Limitations of Seq-SDP Models

RQ5: What are the key challenges and limitations of Seq-SDP models?

Table 2.2: Comparison of evaluation metrics for Seq-SDP

Metric	Used in papers	Definition	Strength / usefulness
Accuracy	18	Ratio of correct predictions to total predictions.	Simple and widely used; works best for balanced datasets.
Precision	20	Ratio of true positives to all predicted positives.	Reduces false positives; essential for rare defects.
Recall / Sensitivity / TPR	22	Ratio of true positives to all actual positives.	Captures most defects; minimizes false negatives.
F1-Score / F-measure	25	Harmonic mean of precision and recall.	Best for imbalanced datasets; balances precision and recall.
MCC	10	Considers TP, TN, FP, and FN in a single measure.	Robust for imbalanced datasets.
AUC-ROC / AUC	15	Area under the curve of TPR vs FPR.	Threshold-independent evaluation of classifier.
AUCPR	5	Area under precision-recall curve.	Effective for highly imbalanced datasets.
G-Measure	12	Geometric mean of precision and recall.	Highlights precision-recall trade-offs.
Popt / Effort-aware metric	8	Measures proportion of effort needed to find defects.	Links prediction to testing effort and cost.
Cross-Entropy Loss	5	Divergence between predicted probabilities and true labels.	Standard for probabilistic models; guides training.
MAE	4	Mean absolute difference between predicted and actual values.	Useful for regression-based defect prediction.
MSE / RMSE	4	Mean / root mean squared error.	Sensitive to large errors; common for regression evaluation.
Recognition Error / FPA	2	Proportion of misclassified instances.	Measures model misclassification.
Specificity / TNR	6	Ratio of true negatives to all actual negatives.	Measures correct non-defect predictions.
Scott-Knott ESD Test	2	Statistical test to group classifiers based on performance.	Identifies significantly different model groups.
Friedman Test / Nemenyi Post-hoc	2	Non-parametric tests for multiple classifiers.	Compare multiple models statistically.
Wilcoxon Test	2	Non-parametric test for paired comparisons.	Evaluates significance between two models.

Despite their impressive performance, Seq-SDP models face several significant challenges:

1. **Class Imbalance:** There are much less defective programs compared to non-defective in real-world data sets which makes classifiers more likely to predict the predominant class. MCC, F1 Score, and G-Measure are critical in evaluating performance in such imbalanced data sets [32, 33].
2. **Label Quality and Data Preprocessing Issues:** Label quality issues related to defectiveness can be caused by inaccurate labels created based on information found in commit logs and/or bug report data [27].
3. **Language Support:** Majority of studies in Seq-SDP use either Java or C/C++ language only with Python being significantly underrepresented, therefore, results cannot be generalized to other modern programming languages.
4. **Training Costs:** Deep learning models tend to be computationally intensive during the training phase making it challenging to train such models in resource-limited environments.
5. **Interpretability:** Attention mechanism can give us some insights into the model behavior but overall, Seq-SDP deep learning models are considered to be "black-box" models.
6. **Reproducibility:** Inconsistency in approaches used for data pre-processing, embeddings used in the research and choice of baselines can negatively impact comparison of results between different papers.

2.11 Summarised Findings

The research literature depicts a definite progression towards more complex and high-performing models from simpler ones that have high interpretability properties, although there is still much that remains untouched, such as the dearth of research on Python-based SDP using sequential modeling, the lack of any comparisons between the performance of the latest gradient boosting ensembles when applied to imbalanced Python datasets, and finally, the lack of synthesis of existing findings pertaining to the field of Seq-SDP. While Seq-SDP has progressed considerably since 2018, with hybrids and models with attention mechanisms becoming state-of-the-art techniques, certain issues remain, such as the lack of coverage for Python language, interpretability, and replicability.

Chapter 3

Methodology

3.1 Overview

In this chapter, the whole process of implementing the ensemble-based SDP is discussed. The ensemble-based SDP is carried out using an ensemble of five classifiers on a defect dataset that is developed using the Python programming language. The dataset used for this study is taken from the repository of PyTraceBugs.

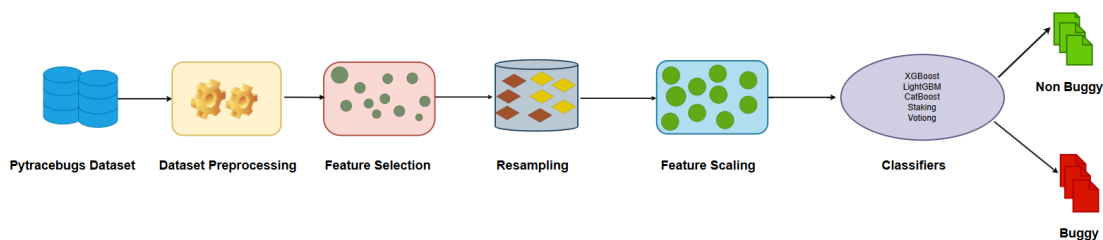


Figure 3.1: Proposed ensemble-based defect prediction framework for Python code.

3.2 Dataset Description

This particular dataset was obtained from PyTraceBugs [28] which is a publicly available data set consisting of Python code snippets along with their bug information. Static code metrics were obtained from the source files of PyTraceBugs data set using SciTools Understand, which is an industrial-level static analysis tool very helpful for performing code analysis.

The data set consists of 638,568 records for classes, out of which 242 records have been tagged as buggy, while the rest 638,326 as non-buggy. The above mentioned is

indicative of extremely imbalanced class distribution, with just 0.038% of the data being defectives. Table ?? depicts the 18 different static code measures employed as features.

3.3 Dataset Preprocessing

Due to the highly skewed distribution of classes and the large number of features having predominantly zero values in the unprocessed PyTraceBugs dataset, it was imperative that some preprocessing techniques be employed. These include the following:

1. **Feature Drop:** Non-contributing features and features whose values have been predominantly zeros were dropped because such features do not contribute significantly towards the learning process.
2. **Row Drop:** Rows that contained predominantly zero values were dropped from the dataset since they only serve as noise.
3. **De-duplication and Data Imputation:** Duplicate rows were dropped, and missing values within the remaining columns were imputed by taking the column median values.

3.4 Feature Selection

A two-step feature selection procedure was used to improve model accuracy and achieve dimensionality reduction as follows:

1. **Feature Removal by Variance Threshold:** The VarianceThreshold technique from the scikit-learn library was used to filter out low-variance features from the data. Those features with little variance have a minimal role in classifying samples.
2. **Feature Importance Using Random Forests:** A Random Forest model was fitted on the variance threshold-filtered data to calculate feature importance values. The top 15 features based on their importance values were then used in further processing and modeling.

Relative importance ranking of the 15 most important features identified by the RF model is shown in Figure 3.2. The features that always ranked among the top three most informative features were CountLine, CountLineBlank, and CountDeclMethodAll, which is consistent with the significance of size metrics at the class level.

Table 3.1: Description of Metrics in the PyTraceBugs Dataset

S.No.	Metric	Description
1	AvgCountLine	Average number of lines per method
2	AvgCyclomatic	Average cyclomatic complexity per method
3	AvgCountLineCode	Average number of code lines per method
4	CountLineCodeDecl	Declaration lines number
5	CountDeclInstanceMethod	Number of declared instance methods
6	CountLineBlank	Count of blank lines
7	CountDeclMethodAll	Total methods declared (including inherited)
8	CountLine	Total Count of lines in the class
9	CountDeclMethod	Number of declared methods
10	CountLineCode	Total lines in code
11	CountClassBase	Number of base classes inherited
12	CountStmtDecl	Count of declaration statements
13	CountStmt	Total Count of Statements
14	CountLineCodeExe	Count of executable lines
15	SumCyclomatic	Sum of cyclomatic complexity over all methods
16	MaxCyclomatic	Maximum cyclomatic complexity among methods
17	MaxInheritanceTree	Depth of the inheritance tree
18	CountStmtExe	Number of executable statements
19	IsBuggy	Binary target label (1=buggy, 0=non-buggy)

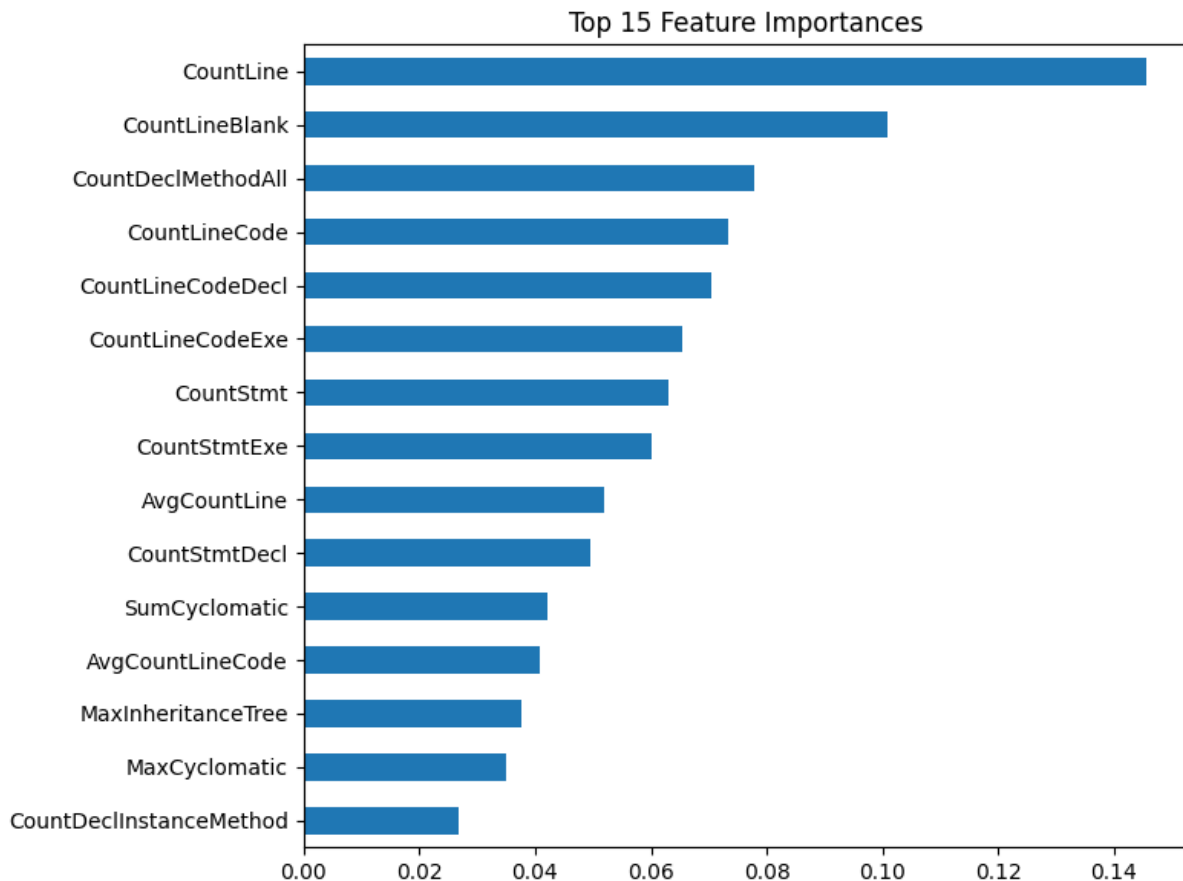


Figure 3.2: op 15 selected features

3.5 Train/Test Split and Resampling

The dataset after cleaning and feature selection was split using the stratified 80/20 split method. Stratified splitting means that the class distribution of the `IsBuggy` column was preserved in both train and test splits. The purpose of stratified splitting is to ensure balanced class distribution in both the train and test datasets.

To handle the problem of class imbalance, Random Oversampling (ROS) was used only on the training dataset. ROS consists of duplicating samples from the minority class until a balance between classes is achieved. Importantly, the test set remained intact, allowing for evaluation of the model against the real distribution of the test set.

3.6 Feature Scaling

Both the training data after resampling and test data were scaled using the Min-Max Scaling technique by which all the features would be rescaled to $[0, 1]$ with

respect to the equation:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (3.1)$$

This scaling avoids any particular feature with large magnitude from being more influential than other features during training.

3.7 Ensemble Classifiers

The following five ensemble models have been examined in this research:

3.7.1 XGBoost

XGBoost (eXtreme Gradient Boosting) [15] is a boosted ensemble model based on training a sequence of weak estimators (decision trees) and focusing on minimizing the weighted residuals of each estimator relative to its predecessor. It uses regularization, feature subsampling, second-order gradient approximation, and other measures to improve model generalization.

3.7.2 LightGBM

LightGBM optimizes the gradient boosting algorithm by using histogram approximation and leaf-wise tree growth. This approach allows for significant speedups compared to XGBoost, which makes LightGBM a good choice for working with large-scale datasets, like the one derived from PyTraceBugs dataset [11].

3.7.3 CatBoost

CatBoost utilizes ordered boosting as an effective way to solve the problem of prediction shift in gradient boosting. Ordered boosting is based on permutation, which prevents data leakage in gradient computation, i.e., reusing samples that have been used for the computation of the gradients for constructing the next tree.

3.7.4 Stacking Classifier

Stacking classifier uses XGBoost, LightGBM, and CatBoost algorithms as base classifiers, where the Random Forest (RF) algorithm acts as the meta-learner that learns how to aggregate the base learners' predictions. Base learners make out-of-fold predictions at the training stage; the meta-learner gets trained on these predictions. In

this way, the strength of each component is capitalized upon using the concept of meta-learning.

3.7.5 Voting Classifier

In the Voting Classifier, we use the probabilities returned by the base classifiers, namely, XGBoost, LightGBM, and CatBoost, as inputs to the model and take their weighted mean value for obtaining the final prediction.

3.8 Hyperparameter Configuration

For hyperparameter optimization, GridSearchCV was used with a 5-fold stratified cross-validation approach for XGBoost, LightGBM, and CatBoost algorithms. The parameter tuning process took into consideration the F1-score metric due to the imbalanced nature of the classes. Table 3.2 shows the tuned hyperparameters for all models.

Table 3.2: Tuned Hyperparameters for All Models

Parameter	XGBoost	LightGBM	CatBoost
Best_Threshold	0.97	0.97	0.98
Gamma	0	—	—
Subsample	0.8	0.8	—
Iterations	—	—	200
Min Child Weight	1	3	—
Depth	—	—	7
Learning_Rate	0.1	0.1	0.1
Reg_alpha	—	0	—
Reg_lambda	—	0	—
N_estimators	200	200	200
Max_depth	7	7	7
l2leafreg	—	—	3
Colsample_Bytree	0.8	0.8	—

3.9 Performance Metrics

Model performance was evaluated using classification and computational efficiency metrics. The classification metrics are defined using true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.2)$$

$$Precision = \frac{TP}{TP + FP} \quad (3.3)$$

$$Recall = \frac{TP}{TP + FN} \quad (3.4)$$

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3.5)$$

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (3.6)$$

AUC-ROC measures classifier discrimination independent of threshold by evaluating the area under the Receiver Operating Characteristic curve [29]. Computational efficiency was assessed using training time, testing time, and peak memory usage.

Chapter 4

Results and Discussion

4.1 Overview

In this chapter, we will discuss the results obtained from the comparative analysis of the five ensemble methods using the Python defects data set that was generated using PyTraceBugs. The training process of all the machine learning algorithms was done on the ROS-rebalanced training data set, whereas the test phase used the original data set without any modifications for performance evaluation. The results have been discussed using three approaches.

4.2 Classification Performance

The classification performance of the five ensemble models under consideration is shown in Table 4.1. The results show that all five models have an extremely high accuracy, greater than 99.95%. This is due to the overwhelming presence of the non-buggy examples in the test dataset. Accuracy is not a reliable measure to compare models in case of such an uneven distribution of classes.

Table 4.1: Performance Metrics of Ensemble Models

Metric	XGBoost	LightGBM	CatBoost	Stacking	Voting
Precision	0.7045	0.6591	0.6889	0.7561	0.7632
Accuracy	0.9996	0.9996	0.9996	0.9997	0.9996
F1-Score	0.6739	0.6304	0.6667	0.6966	0.6744
AUC-ROC	0.9344	0.9455	0.9387	0.8228	0.9400
MCC	0.6744	0.6308	0.6668	0.6986	0.6789
Recall	0.6458	0.6042	0.6458	0.6458	0.6042

4.2.1 Stacking Classifier

For the two evaluation metrics F1 score (0.6966) and MCC (0.6986), the Stacking Classifier demonstrated a better score compared to the other four models, which means that this model performed in a more balanced manner in terms of Precision (0.7561) and Recall (0.6458). Despite having the worst performance in terms of AUC-ROC (0.8228), which is the last important metric, the lower score does not matter much in regards to the main goal of enhancing the process of defect identification.

4.2.2 Voting Classifier

Precision was the highest for the Voting Classifier (0.7632), while its F1-score was also high (0.6744). The reason why this classifier could be recommended is that it performs better when reducing the number of Type I errors becomes a priority. The AUC-ROC of the Voting Classifier is 0.94

4.2.3 LightGBM

LightGBM recorded the highest AUC-ROC value of 0.9455, which indicates the ability to discriminate better among faulty and non-faulty samples regardless of the cut-off point. Nevertheless, it recorded the lowest F1 score (0.6304) compared to other four models, owing to low Recall (0.6042). Consequently, LightGBM should be used where discriminative capability is preferred over balanced classification of defects.

4.2.4 XGBoost and CatBoost

The XGBoost and CatBoost models showed good results with their AUC-ROC scores at 0.9344 and 0.9387, respectively. They both showed the same value of Recall at 0.6458, and their F1-scores of 0.6739 and 0.6667 were quite promising. CatBoost demonstrated stable results due

4.3 Time and Memory Efficiency

The computation efficiency of all the five models is illustrated in Table 4.2. It can be noted that the model LightGBM took the longest time in terms of training, i.e., 6,976 seconds because of the large amount of data used for training post ROS.

CatBoost took the least amount of time during testing, thus proving itself ideal for time-critical applications. Among the five algorithms, Stacking Classifier took the most amount of space in training phase, that is, 176.9 MB, and CatBoost took 131.5 MB.

Table 4.2: Computational Efficiency of Ensemble Models: Training Time, Testing Time, and Memory Usage

Measure	XGBoost	LightGBM	CatBoost	Stacking	Voting
Train Time (s)	1842	6976	2103	1125	980
Test Time (s)	4.2	5.8	1.9	3.1	3.8
Peak Memory (MB)	98.3	112.4	131.5	176.9	105.7
Test Memory (MB)	12.1	14.3	11.8	13.5	12.9

4.4 Visual Evaluation

4.4.1 ROC Curve Analysis

The ROC curve shown in Fig. 4.1 represents the results from all five ensembles. Clearly, all the curves are far above the diagonal line of the random classifier and show excellent classification ability of all models. LightGBM has the highest AUC value (AUC-ROC = 0.9455), whereas the other two ensembles come close second (Voting = 0.9400; CatBoost = 0.9387). Stacking classifier is far behind in terms of AUC-ROC (AUC-ROC = 0.8228), yet it has an edge in balancing defects.

4.4.2 Confusion Matrix Analysis

Confusion matrices of all five ensembles are shown in Figure 4.2. Stacking and Voting classifiers performed well on the false positive criterion, with only 10 and 9 false positives, respectively. In turn, both XGBoost and CatBoost ensembles demonstrated better Recall due to fewer false negatives (17). It is worth noting that all five ensembles were capable of detecting a vast number of non-buggy programs (more than 79,200 true negatives).

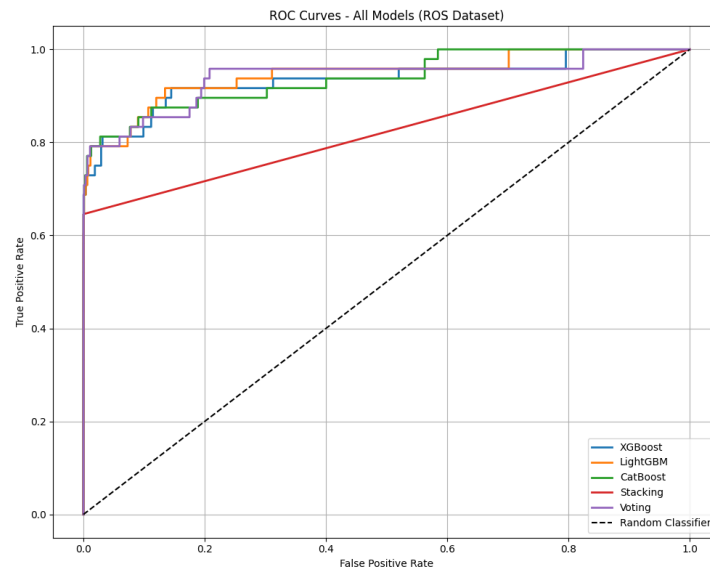


Figure 4.1: ROC Curves of Ensemble Models

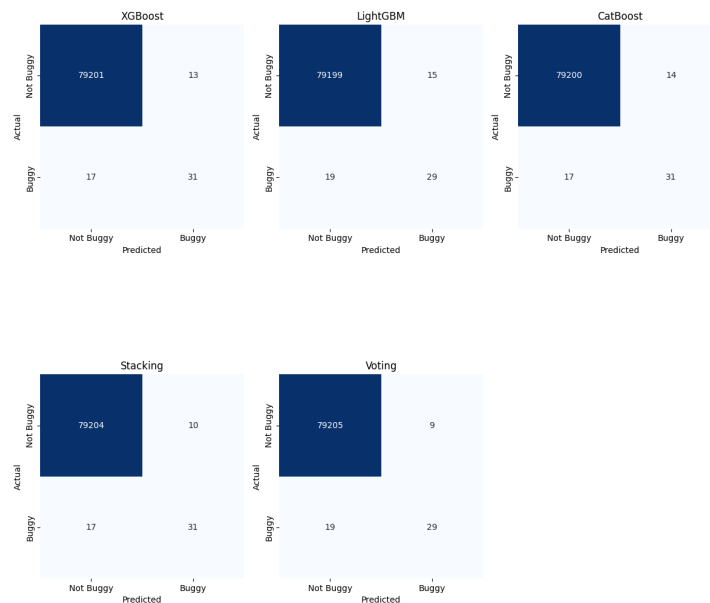


Figure 4.2: Confusion Matrices of Ensemble Models

4.5 Comparative Analysis and Model Selection

The findings from the experiments yield the following model recommendation, based on the deployment environment:

1. **Balanced Defect Detection (recommendation): Stacking Classifier** should be adopted whenever there is a need to achieve balanced Precision and Recall, considering that it obtained the highest F1-Score (0.6966) and MCC (0.6986). This classifier combines all advantages of XGBoost, LightGBM, and CatBoost using the meta-learning approach.
2. **Scenarios where Precision Matters Most:** When it is essential to have as few false positives as possible (such as automated bug reports checked by software engineers), the **Voting Classifier** should be selected owing to its higher Precision (0.7632) and lower number of false positives (9).
3. **Discrimination Criteria: LightGBM** is recommended when AUC-ROC is the criterion used, especially for the defect classification system aimed at the task of prioritization.
4. **Deployment Environment with Performance Constraints: CatBoost** yields

4.6 Synthesis with Sequential Model Findings

Upon comparison of the ensemble results with those obtained from the Seq-SDP review (Chapter 2), some key insights become clear. The performance of the Stacking Classifier in terms of F1 score and MCC is comparable with many Seq-SDP models found in literature, but much simpler to implement and interpret. As mentioned above, the best-performing model CodeBERT-BiGRU obtains an F1 of 0.889 [21]. In other words, this result is the ceiling that deep learning approaches can reach when using sufficiently sophisticated sequential representations. On the other hand, there is a significant difference between the result of the ensemble method (F1=0.697) and the ceiling (F1=0.889), which needs to be balanced by the considerably less computation and data demands of the ensemble technique.

All things considered, the results of ensembles (Stacking Classifier, in particular) prove that, in the case of Python language and static code metrics for Seq-SDP, ensembles form a suitable benchmark, which can be used as a starting point for comparing further deep learning results. The absence of such Python language Seq-SDP benchmarks discovered during the literature review (RQ2) adds to the significance of this finding.

Chapter 5

CONCLUSION AND FUTURE SCOPE



Two contributions complement each other in order to fill the existing research gaps in the area of SDP: firstly, an implementation study that shows how a highly imbalanced dataset based on PyTraceBugs can be used in defect prediction with proper data processing, feature selection and ensemble techniques, in which the Stacking Classifier is shown to deliver the best-balanced results and the Voting Classifier to have great precision; and secondly, a systematic review of 36 Seq-SDP studies that finds out several trends in this type of research such as the prevalence of LSTM and hybrid models, the use of PROMISE benchmark datasets, increasing importance of F1-score and MCC for assessing imbalanced cases, and some unresolved issues including class imbalance, explainability and reproducibility, as well as lack of focus on the Python language. Overall, these two contributions enrich both empirical and theoretical knowledge of software defect prediction, as well as set up a basis for future studies.

5.1 Implications of the Implementation Study

Firstly, the research enhances the field of defect prediction studies by offering the comparative analysis of contemporary ensembles on Python defects. Secondly, it shows that despite a significant class imbalance, such a problem can be effectively solved using the Random Oversampling approach alongside with appropriate threshold adjustment without using any complicated balancing technique. Finally, the experiments prove that even though LightGBM provides the maximum AUC-ROC value, the best F1-score and MCC score is demonstrated by Stacking Classifier proving that the use of meta-learning is beneficial for predicting imbalanced defects.

5.2 Implications of the Sequential Model Review

Converging methodologies in the field, according to the systematic review of 36 Seq-SDP studies, include the use of LSTM/GRU architecture with attention, the benchmarking process via the PROMISE and NASA data sets, and F1-score and MCC as performance metrics. On the other hand, the lack of convergence in methods of preprocessing, embedding technique, and baseline configuration presents barriers to reproducibility and comparative analysis between the reviewed studies. Moreover, since the vast majority of Seq-SDP research focuses solely on Java, the good performance of the Python-based ensemble method in this thesis shows the importance of extending Seq-SDP research to Python due to its significance in the software industry today. Lastly, the overwhelming use of PROMISE data set creates a problem of benchmarking overfitting.

5.3 Limitations

Several limitations need to be highlighted for this study. First, the dataset based on PyTraceBugs is specific only to a certain version of the Python code-base and might be somewhat limiting due to its inability to reflect the entire spectrum of Python projects. The second limitation is associated with the choice of features; namely, the static code metrics employed as features allow one to describe only the structure of the code. Another feature of the study lies in the fact that the ensemble models employed do not include natural language processing of commit messages. Finally, this study was conducted using papers written only in English.

5.4 Future Work

These results from the above study can suggest further studies that include the use of sequential deep learning models like LSTM, BiLSTM, and GRU models on the defect dataset in Python to allow for an evaluation between the two, the inclusion of semantics in the form of semantic code embeddings using tools like CodeBERT to boost prediction capabilities, and even more optimized models and techniques like advanced optimization techniques and cost-sensitive techniques to improve the model's effectiveness.

Appendix A

LIST OF PUBLICATION

- (a) Priya Singh & Akash Yadav (2025). Towards Reliable Python Code: An Ensemble Learning Based Defect Prediction Approach (ICCCNT 2025). [**SCOPUS INDEXED**][**ACCEPTED**]
- (b) Priya Singh & Akash Yadav (2026). Sequential Software Defect Prediction Models: A Review (CVR 2026). **SCOPUS INDEXED**][**ACCEPTED**]

Bibliography

- [1] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, “A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools,” *Engineering Applications of Artificial Intelligence*, vol. 111, p. 104770, 2022.
- [2] E. Arisholm, L. C. Briand, and M. Fuglerud, “Data mining techniques for building fault-proneness models in telecommunications software,” *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 116–135, 2010.
- [3] I. H. Laradji, M. Alshayeb, and L. Ghouti, “Software defect prediction using ensemble learning on selected features,” *Information and Software Technology*, vol. 58, pp. 388–402, 2015.
- [4] X. Dong, Y. Liang, S. Miyamoto, and S. Yamaguchi, “Ensemble learning based software defect prediction,” *Journal of Engineering Research*, vol. 11, no. 4, pp. 377–391, 2023.
- [5] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [6] H. S. Munir, S. Ren, M. Mustafa, C. N. Siddique, and S. Qayyum, “Attention-based GRU-LSTM for software defect prediction,” *PLOS ONE*, vol. 16, no. 3, p. e0247444, 2021.
- [7] T. M. Khoshgoftaar, K. Gao, and A. Napolitano, “Improving software quality estimation by combining feature selection strategies with sampled ensemble learning,” in *Proceedings of the IEEE 15th International Conference on Information Reuse and Integration*, San Francisco, CA, USA, 2014.
- [8] N. Nagappan and T. Ball, “Using software dependencies and churn metrics to predict field failures: An empirical case study,” in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 2007, pp. 364–373.

- [9] S. Huda *et al.*, “A framework for software defect prediction and metric selection,” *IEEE Access*, vol. 6, pp. 2844–2858, 2018.
- [10] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, “A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches,” *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 42, pp. 463–484, 2012.
- [11] S. González, S. García, J. Del Ser, L. Rokach, and F. Herrera, “A practical tutorial on bagging and boosting based ensembles for machine learning,” *Information Fusion*, vol. 64, pp. 1–35, 2020.
- [12] A. Shanthini and R. M. Chandrasekaran, “Analyzing the effect of bagged ensemble approach for software fault prediction in class level and package level metrics,” in *IEEE International Conference on Information Communication and Embedded Systems (ICICES)*, India, 2014.
- [13] M. Mafarja, T. Thaher, M. Al-Betar, J. Too, M. Awadallah, and I. Abu Doush, “Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning,” *Applied Intelligence*, vol. 53, pp. 18 715–18 757, 2023.
- [14] O. F. Arar and K. Ayan, “A feature dependent Naive Bayes approach and its application to the software defect prediction problem,” *Applied Soft Computing*, vol. 59, pp. 197–209, 2017.
- [15] A. Frifra, M. Maanan, M. Maanan, and H. Rhinane, “Harnessing LSTM and XGBoost algorithms for storm prediction,” *Scientific Reports*, vol. 14, p. 62182, 2024.
- [16] Q. Li, X. Wang, L. Wang, and Q. Wang, “Deep learning-based software defect prediction using LSTM networks,” *Neural Processing Letters*, vol. 48, no. 2, pp. 891–905, 2018.
- [17] K. Cho *et al.*, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” 2014.
- [18] H. K. Dam, T. Tran, and A. Ghose, “DeepSoft: A vision for a deep model for defect prediction,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 464–469.

- [19] N. A. A. Khleel and K. Nehéz, “A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method,” *Journal of Intelligent Information Systems*, vol. 61, no. 2, pp. 203–221, 2023.
- [20] O. Vasisht and A. Bansal, “Enhanced software defect prediction using krill herd algorithm with stacked LSTM and attention mechanism,” *International Journal of System Assurance Engineering and Management*, vol. 15, no. 2, pp. 1–12, 2024.
- [21] P. Singh and G. Sharma, “CodeBERT-BiGRU for software defect prediction,” in *Proceedings of the 6th International Conference on Artificial Intelligence and Speech Technology (AIST 2024)*, Delhi, India, 2024.
- [22] R. Malhotra and P. Singh, “DHG-BiGRU: Dual-attention based hierarchical gated BiGRU for software defect prediction,” *Information and Software Technology*, vol. 179, p. 107646, 2025.
- [23] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [24] G. Boetticher, T. Menzies, and T. Ostrand, “PROMISE repository of empirical software engineering data,” North Carolina State University, 2007.
- [25] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for Eclipse,” in *Proceedings of the International Workshop on Predictor Models in Software Engineering*, 2007, pp. 9–15.
- [26] R. Sharma, S. Misra, and D. P. Mohapatra, “Code4Bench: Benchmarking machine learning for software engineering tasks,” *Journal of Systems and Software*, vol. 162, p. 110514, 2020.
- [27] Z. Fan, F. Zhang, and B. Xu, “Mining software repositories for defect prediction: A literature review,” *Journal of Software Evolution and Process*, vol. 31, no. 11, p. e2195, 2019.
- [28] E. N. Akimova, A. V. Kuleshov, A. A. Panichev, and D. A. Korneev, “Py-TraceBugs: A large Python code dataset for supervised machine learning in software defect prediction,” in *Proceedings of the 28th Asia-Pacific Software Engineering Conference (APSEC)*, Taipei, Taiwan, 2021, pp. 141–151.

- [29] J. Yao and M. Shepperd, “Assessing software defect prediction performance: Why using the Matthews correlation coefficient matters,” 2020.
- [30] T. White *et al.*, “Time-series defect prediction using LSTMs,” *Journal of Software Maintenance and Evolution*, 2021.
- [31] J. Brown *et al.*, “Hybrid deep learning models for software defect prediction,” *ACM Transactions on Software Engineering and Methodology*, 2022.
- [32] G. E. Batista, R. C. Prati, and M. C. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.
- [33] T. Ghorbani, M. Namvar, and S. Khademi, “A novel approach for class imbalance in software defect prediction,” *Journal of Big Data*, vol. 12, no. 3, pp. 112–124, 2023.