HIMANSHU KUMAR

ThesisByHIM

M.TECH. THESIS 2

Document Details

Submission ID
trn:oid:::27535:141735382

Submission Date
Jun 5, 2026, 1:43 AM GMT+5:30

Download Date
Jun 10, 2026, 12:02 PM GMT+5:30

File Name
ThesisByHIM.pdf

File Size
1.5 MB

31 Pages

6,154 Words

38,390 Characters

9% Overall Similarity

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

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text
- Small Matches (less than 10 words)

Match Groups

35 Not Cited or Quoted 9%
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

7% Internet sources
4% Publications
6% Submitted works (Student Papers)

Match Groups

- **35** Not Cited or Quoted 9%
  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

7% Internet sources
4% Publications
6% 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	ebin.pub <1%
2	Internet	odr.chalmers.se <1%
3	Student papers	University of Ulster on 2026-04-20 <1%
4	Student papers	University of Newcastle upon Tyne on 2026-05-05 <1%
5	Internet	ojs.academypublisher.com <1%
6	Publication	Yu Tang, Yang Zhou, Cheng Yang, Ye Du, Ming-song Yang. "Instance gravity overs... <1%
7	Internet	www.researchgate.net <1%
8	Student papers	BITS, Pilani-Dubai on 2021-01-03 <1%
9	Student papers	Delhi Technological University on 2026-05-26 <1%
10	Internet	ijadis.org <1%

11 Student papers
Asian Institute of Technology on 2007-05-15 <1%

12 Student papers
Vellore Institute of Technology on 2026-05-24 <1%

13 Internet
deepai.org <1%

14 Internet
repository.lib.ncsu.edu <1%

15 Internet
www.hindawi.com <1%

16 Publication
Duksan Ryu, Jong-In Jang, Jongmoon Baik. "A Hybrid Instance Selection Using Ne... <1%

17 Student papers
Liverpool John Moores University on 2025-08-05 <1%

18 Internet
nag-9-s.gitbook.io <1%

19 Internet
www.diva-portal.org <1%

20 Internet
impactfactor.org <1%

21 Internet
link.springer.com <1%

22 Internet
www.doria.fi <1%

23 Publication
Amina Adadi, Afaf Bouhoute. "Explainable Artificial Intelligence for Intelligent Tra... <1%

24 Internet
monami.hs-mittweida.de <1%

25 Internet

www.ijcaonline.org <1%

26 Student papers

Liverpool John Moores University on 2023-10-23 <1%

27 Student papers

Liverpool John Moores University on 2026-03-26 <1%

28 Student papers

Middlesex University on 2026-03-12 <1%

29 Student papers

University College London on 2018-04-06 <1%

30 Internet

docs.neu.edu.tr <1%

31 Internet

dokumen.pub <1%

32 Internet

dr.ntu.edu.sg <1%

33 Internet

www.mdpi.com <1%

ABSTRACT

21 Software Defect Prediction (SDP) is an important task to improve the software quality by identifying defect-prone modules in early stages of software development. Accurate prediction of software defects is useful in reducing testing effort, maintenance cost and system failures in large scale software projects. However, current SDP approaches still suffer from challenges such as high dimensional feature spaces, class imbalance and low interpretability of the prediction models.

In this thesis, we propose a holistic framework for software defect prediction by using optimization-based ensemble learning and explainable transformer models. The first part of the work proposes a machine learning framework driven by optimization. It combines Binary Particle Swarm Optimization (PSO) for feature selection, SMOTE-ENN for imbalance handling, Bayesian hyperparameter optimization by Optuna and a soft-voting LightGBM-XGBoost ensemble classifier with threshold calibration. The proposed method improves the classification performance and significantly reduces the feature dimensionality.

The second part of the thesis focuses on transformer based deep learning models for software defect prediction. We propose a hybrid transformer framework to capture complex feature interactions and improve the performance of defect classification. Further, explainability techniques based on SHAP are integrated to enhance model interpretability and provide insights into the feature contributions behind prediction decisions.

Experimental studies on PROMISE datasets show that the proposed methods perform well in terms of F1-score, ROC-AUC and are also computationally efficient. The proposed framework helps to build reliable, interpretable and scalable software defect prediction systems for today's software engineering practices.

Contents

Acknowledgement i

Candidate's Declaration ii

Certificate iii

Abstract iv

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Background and Motivation 1
1.2 Problem Definition 2
1.3 Research Objectives 2
1.4 Research Contributions 3
1.5 Organization of the Thesis 3

2 Technical Background 5

2.1 Software Defect Prediction 5
2.2 Feature Selection Techniques 7
2.3 Ensemble Learning 7
2.4 Transformer-Based Deep Learning 7
2.5 Explainable Artificial Intelligence 8
2.6 Evaluation Metrics 8

3 Literature Review 9

3.1 Ensemble Learning Based SDP Approaches 10
3.2 Imbalance Handling Techniques 10
3.3 Optimization-Based Feature Selection 10
3.4 Deep Learning-Based SDP Models 11
3.5 Explainable AI in SDP 12

3.6 Research Gaps 12

8 **4 Proposed Methodology** **13**

4.1 Dataset Description 13
4.2 Data Preprocessing 13
4.3 Binary PSO-Based Feature Selection 13
4.4 SMOTE-ENN Resampling 14
4.5 Optimization-Based Ensemble Framework 14
4.5.1 LightGBM 14
4.5.2 XGBoost 15
4.5.3 Soft Voting Ensemble 15
4.6 Optuna-Based Hyperparameter Optimization 15
4.7 Threshold Calibration 15
4.8 Transformer-Based Deep Learning Framework 15
4.9 SHAP Explainability Analysis 16

5 Experiment Results and Discussion **18**

5.1 Experiment Setting 18
5.2 Performance Measures 18
5.3 Optimization-Based Framework Results 19
5.4 Transformer-Based Framework Results 19
5.5 ROC Curve Analysis 19
5.6 Ablation Study 20
5.7 Feature Reduction Analysis 20
5.8 Threshold Analysis 20
5.9 Comparative Analysis 21
11 5.10 Key Findings and Limitations 21

6 Conclusion **23**

6.1 Conclusion 23
6.2 Future Directions 24

List of Tables

2.1 Major Machine Learning Algorithms Used in SDP 6
2.2 Common Imbalance Handling Techniques in SDP 6
3.1 Traditional Machine Learning Techniques Used for SDP 10
4.1 NASA PROMISE Datasets Used in the Proposed Framework 13
5.1 Configuration settings for the experiment 18
5.2 Performance of optimization-based ensemble framework 19
5.3 Feature Selection Efficiency 21
5.4 Comparative Analysis of Proposed Frameworks 21

List of Figures

4.1 Threshold calibration analysis for different datasets. 16
4.2 SHAP-based explainability workflow. 17

5.1 Comparative performance of transformer-based framework 19
5.2 ROC curve analysis for optimization-based framework 20
5.3 Ablation study showing impact of components on F1-score 21

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Modern computing technologies, cloud computing, mobile apps, and enterprise systems with a significant scale have increasingly led to the proliferation of software systems in their complexity. Therefore, one of the main problems in software engineering is to ensure software reliability and quality. A software defect during development can lead to failure of the software, higher maintenance expenses, dissatisfied customers, and security weaknesses. The early detection of software defect is crucial in enhancing software quality and minimizing software maintenance effort.

Software Defect Prediction (SDP) is an important research area which aims to select potentially defect software modules from the historical software metrics and intelligent prediction techniques. With accurate defect prediction, software teams can prioritize their testing efforts, allocate resources effectively, and enhance software reliability. Typical software defect prediction methods include the usage of software metrics like cyclomatic complexity, coupling metrics, cohesion metrics, Halstead metrics and lines of code from software repositories.

Several machine learning models like Logistic Regression, Decision Trees, random forest, support vector machine, and ensemble learning models have been extensively investigated for software defect prediction problems. While these methods have yielded promising results, they suffer from a number of problems such as high dimensional feature space, redundant software metrics, strong class imbalance and hyperparameter sensitivity.

With the recent developments of deep learning, software defect prediction models have been further enhanced by using auto-learning to detect the complex feature representations. Transformer-based architectures have shown excellent performance to learn long-range feature dependencies with self-attention mechanisms. But deep learning models often operate as a black box with less interpretability, leaving the results of their predictions hard to analyse in real software engineering applications.

The imbalance between the number of defective and non-defective software modules is an important problem in software defect prediction. Traditional classifiers are biased towards the majority classes and have poor detection of the minority defective modules.

The classification performance will also be adversely affected by the poor feature selection and sub-optimal threshold selection.

Motivated by these problems, this thesis proposes a comprehensive framework to tackle the software defect prediction problem using optimizing-based ensemble learning and explainable Transformer models. The proposed work presents a predictive model that combines feature selection, imbalance handling, ensemble learning, deep learning, hyper-parameter optimization and EAI techniques.

1.2 Problem Definition

Although there are significant advances in the software defect prediction research, there are still important challenges with the existing techniques. The datasets of the software defects are often highly imbalanced with more than a thousand software metrics. The traditional machine learning models often perform poorly due to some redundant software metrics. Most of the studies focus on optimizing individual components, like feature selection or training classifiers, but not on forming a complete optimization-based prediction system.

Although deep learning methods have enhanced the representation learning ability for software defect prediction, most deep learning models are not interpretable or transparent. Because of the black-box approach of deep learning systems, their trustworthiness is limited and their applicability in practical use in a real software engineering environment.

Moreover, very often, the prediction performance for the minority classes is not good because of improper threshold selection and also because of ineffective imbalance handling techniques. Hence, an integrated software defect prediction framework is needed that can enhance the predictive performance whilst retaining interpretability, robustness and computational efficiency.

1.3 Research Objectives

The aims of this thesis are as follows:

1. To design an optimization based ensemble learning for software defect prediction with the help of feature selection, imbalance handling and hyperparameter optimization techniques.

2. To enhance the software defect prediction performance using Transformer-based deep learning architectures that can learn complex feature interactions.

3. To incorporate EAI methods into enhancing the interpretability and understanding of the contributions of features to the decisions made in a prediction task.

4. To assess the effectiveness of the proposed approaches with the popular software defect prediction datasets and common evaluation metrics.

1.4 Research Contributions

The major contributions of this thesis are summarized as follows:

1. Development of an optimization-driven ensemble framework integrating Binary Particle Swarm Optimization (PSO), SMOTE-ENN resampling, Optuna-based hyperparameter optimization, and LightGBM-XGBoost soft voting ensemble learning for software defect prediction.

2. Development of a Transformer-based deep learning framework incorporating convolutional feature extraction, positional encoding, focal loss, and self-attention mechanisms for improved software defect prediction.

3. Integration of SHAP-based explainability techniques to improve interpretability and transparency of software defect prediction models.

4. Comprehensive experimental evaluation and comparative analysis using benchmark PROMISE software defect prediction datasets.

1.5 Organization of the Thesis

The following outline provides some organization of the thesis.

The other chapters of the thesis are arranged as follows:

Chapter 2 introduces the technical background concerning software defect prediction, software metrics, machine learning methods, handling class imbalance, ensemble learning, transformer-based deep learning, explainable AI and evaluation metrics.

The existing literature about the traditional machine learning approaches, ensemble-based learning methods, imbalance handling techniques, optimization-based feature selection, deep learning approaches, Transformer-based models and explainable artificial intelligence techniques for software defect prediction is discussed in Chapter 3.

The proposed methodology is described in Chapter 4 along with Binary PSO-based feature selection, SMOTE-ENN resampling, optimization-based ensemble learning, Optuna hyperparameter optimization, threshold calibration, Transformer-based deep learning framework, and SHAP explainability analysis.

The proposed frameworks are presented in Chapter 5, where their performance is evaluated by experiments, compared, studied with ablation, reduced in features, analyzed in threshold and also analyzed in computational cost.

Last but not least, the thesis is concluded and research directions for the future in software defect prediction are discussed in Chapter 6.

CHAPTER 2

TECHNICAL BACKGROUND

2.1 Software Defect Prediction

Software Defect Prediction (SDP) is an important area of research in software engineering which aims to predict the defect-prone software modules in a software development lifecycle (SDLC). By predicting software defects early, software organizations can benefit by enhancing software quality, reducing maintenance requirements, optimizing software testing resources, and minimizing the overall development cost.

A number of benchmark repositories like NASA PROMISE repository have been widely adopted to test software defect prediction models due to their availability of standardized datasets that include software metrics and defect labels [1]. In addition, cross-project and within-project software defect prediction approaches have been broadly studied, as they are designed to enhance the generalizability of defect prediction in different software systems [2].

The software metrics used for defect prediction will be discussed.The software metrics for defect prediction will be presented.

Software metrics are quantitative measures that quantify various aspects of software systems. The metrics are used in software defect prediction models as input features and used to find out patterns of defect-prone modules.

In SDP, the following are some common software metrics:

- Lines of Code (LOC) .
- Complexity Metrics: Cyclomatic complexity metrics are used to measure the complexity of decisions in programs.
- Object-oriented design quality is assessed by Object Oriented Metrics such as Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT) and Lack of Cohesion in Methods (LCOM).
- Halstead Metrics: Metrics based on operators and operands in the source code.

While traditional software metrics exhibit good predictive power, metrics designed manually often do not preserve more semantic relationships within software systems [3].

In this section, we discuss machine learning approaches for SDP.In this section, we look at machine learning approaches to SDP.

The reason for using machine learning techniques for software defect prediction is that they can learn the relationship between software metrics and defect labels. Logistic Regression, Naïve Bayes, Support Vector Machines (SVM), Decision Trees and Random Forests are some of the traditional supervised learning algorithms that have shown to be good predictive models [4].

Furthermore, ensemble learning techniques can be used to boost the prediction power by associating several classifiers to diminish overfitting and enhance the prediction robustness. There are efficient implementations of many machine learning algorithms used in SDP research, available on the libraries like scikit-learn [5].

Table 2.1. Major Machine Learning Algorithms Used in SDP

Algorithm	Category	Characteristics
Logistic Regression	Linear Classifier	Simple and interpretable
Decision Tree	Rule Based Learning	Easy visualization
Random Forest	Ensemble Learning	Robust against overfitting
SVM	Margin-Based Learning	Effective in high dimensions
XGBoost	Gradient Boosting	Strong predictive performance
LightGBM	Gradient Boosting	Computational efficiency

17 Class Imbalance in SDP.

Class imbalance is one of the most crucial problems in software defect prediction as the number of defective software modules are in small proportion to the number of total software instances.

Several oversampling and undersampling techniques have been suggested to overcome this problem. Synthetic Minority Oversampling Technique (SMOTE) creates synthetic minority class samples from the neighbouring minority class samples [6].

$$x_{new} = x + \delta(x_{NN} - x) \tag{2.1}$$

Here x_{NN} denotes a "nearest neighbor" sample and δ is a random number from 0 to 1.

Table 2.2. Common Imbalance Handling Techniques in SDP

Method	Category	Purpose
SMOTE	Oversampling	Generate minority samples
Random Undersampling	Undersampling	Reduce majority dominance
ENN	Noise Removal	Avoid ambiguous samples
SMOTEENN	Hybrid Method	Balance and clean datasets

2.2 Feature Selection Techniques

In practice, software defect datasets often consist of redundant and highly correlated software measurements which have an adverse impact on the accuracy of the classification process and exacerbate the computational complexity.

Three different types of feature selection methods are typically used: filter, wrapper, and embedded. The evolutionary optimization methods like Particle Swarm Optimization (PSO) have shown great potential in exploring high dimensional feature spaces [7].

In Binary Particle Swarm Optimization, the feature subsets are represented in binary vectors, and the particles' positions are updated iteratively.

$$v_{id}^{(t+1)} = wv_{id}^{(t)} + c_1 r_1 (p_{id}^{best} - x_{id}^{(t)}) + c_2 r_2 (g_d^{best} - x_{id}^{(t)}) \tag{2.2}$$

2.3 Ensemble Learning

The ensemble method uses multiple different classifiers to enhance the accuracy and stability of the model.

These non-linear interaction between features are often complex and hard to model and are useful for software defect prediction tasks. This is why gradient boosting frameworks like XGBoost and LightGBM have gained popularity for these purposes. [8, 9].

XGBoost applies regularization and tree pruning algorithms and LightGBM uses histogram-based algorithms and leaf-wise algorithms to grow trees in order to improve the efficiency of the training process.

A soft voting ensemble which combines prediction probabilities from several classifiers, and often outperforms individual classifiers.

2.4 Transformer-Based Deep Learning

The software defect prediction problem has inspired the use of deep learning for learning hierarchical representation of features automatically from software data, which has attracted much attention.

The transformer architecture were first used in natural language processing and then extended to software engineering applications due to its nature of capturing long-range feature dependencies through the self-attention mechanisms from the NLP field [10].

In contrast to recurrent neural networks, Transformer models operate on a parallel fashion, and are efficient in capturing interactions between local and global features.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

Adding convolutional layers along with Transformer encoders, hybrid architectures have been proposed to boost the representation learning ability of the system by modeling local feature patterns before the global attention-based modeling, as proposed in [11].

2.5 Explainable Artificial Intelligence

Deep learning models are often opaque, or not very interpretable. Explainable Artificial Intelligence (XAI) techniques enhance transparency by providing explanations of decisions made during prediction using machine learning and deep learning algorithms.

SHAP (SHapley Additive exPlanations) is founded on cooperative game theory and determines the contribution value of each feature [12].

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i \tag{2.4}$$

In the above, the base prediction value is represented by ϕ_0 and the contribution of feature i is represented by ϕ_i.

With SHAP-based explainability, developers can gain insight into which software modules are predicted as defective.

2.6 Evaluation Metrics

Often software data sets are imbalanced, which is why performance evaluation is an important part of software defect prediction.

Common measures of evaluation include Accuracy, Precision, Recall, F1-score, Matthews Correlation Coefficient (MCC), and ROC-AUC [13].

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.5}$$

ROC-AUC is a metric of the classifier discriminative power at different thresholds.

CHAPTER 3

LITERATURE REVIEW

Software Defect Prediction (SDP) has become an important research area in software engineering to identify the defect prone modules before software deployment. Many studies have been performed on the machine learning and deep learning techniques to improve the prediction performance and the reliability of the software. This chapter provides a review of the major research contributions related to the proposed work.

The study of Software Defect Prediction (SDP) has been extensively studied with the traditional machine learning techniques due to their simplicity, interpretability and computational efficiency.

Hall et al. performed a systematic review of software fault prediction models in which they found that there is no single model that consistently outperforms the others across all of the data sets [4].

Malhotra has written a comprehensive survey on machine learning methods for software fault prediction and discussed the significance of feature engineering and imbalance handling to increase the performance of the classifiers [14].

For SDP tasks, several traditional classifiers have been widely used such as Logistic Regression, Naïve Bayes, Support Vector Machines (SVM), Decision Trees, and Random Forests.

Multiple classification methods were compared and it was shown that ensemble-based classifiers often outperform individual learning methods by Lessmann et al. [15].

Software defect prediction, in particular across projects, has also been given much attention due to its applicability to industrial software systems. Pal and Sillitti have explored several defect prediction approaches across projects and noted that cross-project defect prediction problems still suffer from significant challenges of cross-project heterogeneity of datasets and feature inconsistency [2].

Malhotra and Bansal investigated the strength of process metrics and static code metrics using the Support Vector Machines (SVM) technique [16].

Table 3.1. Traditional Machine Learning Techniques Used for SDP

Technique	Advantages	Limitations
Logistic Regression	Interpretable and simple	Limited non-linear learning
Decision Tree	Easy visualization	High overfitting tendency
SVM	Effective in high dimensions	Sensitive to kernel selection
Random Forest	Strong ensemble learning	Higher computational cost
Naïve Bayes	Fast training	Strong independence assumption

3.1 Ensemble Learning Based SDP Approaches

The learning techniques that are the ensemble learning techniques enhance the performance of the classification by using the predictions of multiple learning models.

Recently, gradient boosting methods like XGBoost and LightGBM have proven to be effective for software defect prediction [8, 9].

Several recent studies have shown that ensemble learning frameworks outperform single classifiers, especially when the classification is imbalanced, in terms of F1-score and ROC-AUC.

Although many of the methods based on ensembles have performed well, their parameters are optimized at the level of the classifiers, and they do not integrate the feature optimization and threshold calibration mechanisms in a common manner.

3.2 Imbalance Handling Techniques

One of the big problems in software defect prediction is that the number of defective software modules is usually very small as compared to the number of defect-free modules.

He and Garcia led a lengthy discussion on problems with learning from imbalanced data [6].

Jude and Uddin combined SMOTE with explainable machine learning methods and obtained better defect classification results on imbalanced datasets [17].

Begum et al. showed that the integration of machine learning with XAI leads to a better defect identifying ability in software [13].

While handling the imbalance makes classification more accurate, if resampling is not used correctly prior to the cross-validation, it can cause data leakage and inflated performance results.

3.3 Optimization-Based Feature Selection

Feature selection is an important pre-processing step since software datasets often have redundant and highly correlated software metrics present.

Feature selection problems have been extensively investigated with the use of meta-

heuristic optimization algorithms like Genetic Algorithms, Ant Colony Optimization and Particle Swarm Optimization.

Kennedy and Eberhart are the original creators of Particle Swarm Optimization, which was inspired by the principles of swarm intelligence [7].

$$v_{id}^{(t+1)} = wv_{id}^{(t)} + c_1 r_1 (p_{id}^{best} - x_{id}^{(t)}) + c_2 r_2 (g_d^{best} - x_{id}^{(t)}) \tag{3.1}$$

Recent research shows the combined use of optimization-based feature selection with ensemble learning can significantly enhance software defect prediction performance.

3.4 Deep Learning-Based SDP Models

Recently, deep learning techniques have become a focus of much attention due to their capability of learning hierarchical representations of features from software data automatically.

For the SDP tasks, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Gated Recurrent Units (GRUs) have been proposed.

In order to achieve this, Abdu et al. came up with a hybrid deep learning system that combines semantic and traditional software features [3].

Mishra and Sharma suggested a deep learning methodology to predict software defects for continuous integration and continuous delivery software systems [11].

Deep learning algorithms have been shown to provide good predictive accuracy but often they need more data and computational power.

In this section we'll move on to transformer and attention-based models.Now we will learn about transformer and attention-based models.

Self-attention mechanisms have led to the development of transformer architectures that have become highly effective deep learning models, capturing long-range feature dependencies.

Zhang and Wu suggested a Transformer-based framework for software defect prediction [10].

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

Mohapatra et al. introduced tabular software defect prediction architecture called SAINT [18].

Malhotra and Patidar investigated the advanced language models and balanced learning approaches for software change proneness prediction [19].

While the models are very good at predictions, they often can be viewed as black-box

systems with limited interpretability.

3.5 Explainable AI in SDP

Explainable Artificial Intelligence (XAI) techniques enhance understanding and explainability of machine learning and deep learning systems.

Santos et al. studied explainable machine learning techniques on software defect prediction [20].

The SHAP technique is one of the most popular explainability techniques since it offers local and global explanations based on the concepts of cooperative game theory.

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i \tag{3.3}$$

Al-Smadi et al. have suggested interpretable machine learning models using support vector machines (SVM) and deep learning models (CNN) for reliable software defect prediction [12].

Explainable AI techniques assist software engineers in pinpointing vital software metrics that are useful in making a decision on defect prediction.

3.6 Research Gaps

From the literature reviewed, the following are some important research gaps identified:

1. Many traditional machine learning methods struggle to understand the complex semantics and context in software systems.

2. Some software defect prediction studies have focused on optimizing individual parts of the defect prediction process, like feature selection or handling data imbalance, rather than creating whole prediction systems.

3. Deep learning techniques enhance the representation learning function but are generally non-interpretable and non-transparent.

4. Current transformer-based models are still not very easy to explain and practically interpret.

5. A few studies have combined optimization-based ensemble learning, Transformer-based deep learning, balancing, thresholding and explainable AI into a unified framework.

CHAPTER 4

PROPOSED METHODOLOGY

4.1 Dataset Description

The proposed software defect prediction frameworks are evaluated using benchmark datasets obtained from the NASA PROMISE repository. The datasets include CM1, KC1, and PC1 projects, which are widely used for evaluating software defect prediction models.

Each dataset consists of software modules represented using static software metrics along with binary defect labels indicating whether a module is defective or non-defective.

Table 4.1. NASA PROMISE Datasets Used in the Proposed Framework

Dataset	Instances	Features	Defect Ratio
CM1	498	21	9.8%
KC1	2109	21	15.5%
PC1	1109	21	6.9%

4.2 Data Preprocessing

The preprocessing stage prepares the datasets for machine learning and deep learning model training. Missing values and inconsistent entries are removed during preprocessing.

The target labels are converted into binary format where defective modules are represented as 1 and non-defective modules are represented as 0.

Feature normalization is performed using Min-Max scaling:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

Stratified train-test splitting is performed to preserve class distribution across training and testing sets.

4.3 Binary PSO-Based Feature Selection

Binary Particle Swarm Optimization (PSO) is utilized for wrapper-based feature selection to reduce feature redundancy and improve classification performance.

The velocity update equation is represented as:

$$v_{id}^{(t+1)} = wv_{id}^{(t)} + c_1 r_1 (p_{id}^{best} - x_{id}^{(t)}) + c_2 r_2 (g_d^{best} - x_{id}^{(t)}) \tag{4.2}$$

The sigmoid activation function is defined as:

$$S(v_{id}) = \frac{1}{1 + e^{-v_{id}}} \tag{4.3}$$

The fitness function is computed using F1-score performance:

$$Fitness = 1 - F1Score \tag{4.4}$$

4.4 SMOTE-ENN Resampling

Class imbalance is handled using the SMOTE-ENN hybrid resampling technique. SMOTE generates synthetic minority-class samples, while Edited Nearest Neighbour removes noisy instances near class boundaries.

Synthetic minority samples are generated using:

$$x_{new} = x_i + \delta(x_{NN} - x_i) \tag{4.5}$$

where:

- x_i represents minority-class samples,
- x_{NN} represents nearest-neighbor samples,
- δ represents a random value between 0 and 1.

4.5 Optimization-Based Ensemble Framework

The first proposed framework integrates feature selection, imbalance handling, hyperparameter optimization, ensemble learning, and threshold calibration within a unified prediction pipeline.

The framework combines LightGBM and XGBoost classifiers using soft voting ensemble learning.

4.5.1 LightGBM

LightGBM is utilized because of its computational efficiency and ability to capture nonlinear feature interactions.

4.5.2 XGBoost

XGBoost is utilized because of its regularization capability and gradient boosting mechanism.

The objective function is defined as:

$$Obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{4.6}$$

4.5.3 Soft Voting Ensemble

Prediction probabilities obtained from LightGBM and XGBoost are combined using soft voting:

$$P(y = 1|x) = \frac{1}{2}[P_{LGB}(y = 1|x) + P_{XGB}(y = 1|x)] \tag{4.7}$$

4.6 Optuna-Based Hyperparameter Optimization

Optuna-based Bayesian optimization is utilized to identify optimal hyperparameter configurations.

$$\theta^* = \arg\max_{\theta} F1(\theta) \tag{4.8}$$

4.7 Threshold Calibration

Threshold calibration is performed by evaluating multiple threshold values and selecting the threshold maximizing F1-score performance.

4.8 Transformer-Based Deep Learning Framework

The second proposed framework integrates convolutional feature extraction, Transformer encoder architecture, focal loss, and SHAP explainability analysis.

The self-attention mechanism is represented as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.9}$$

Focal loss is utilized for minority-class learning:

$$FL(p_t) = -(1 - p_t)^{\gamma} \log(p_t) \tag{4.10}$$

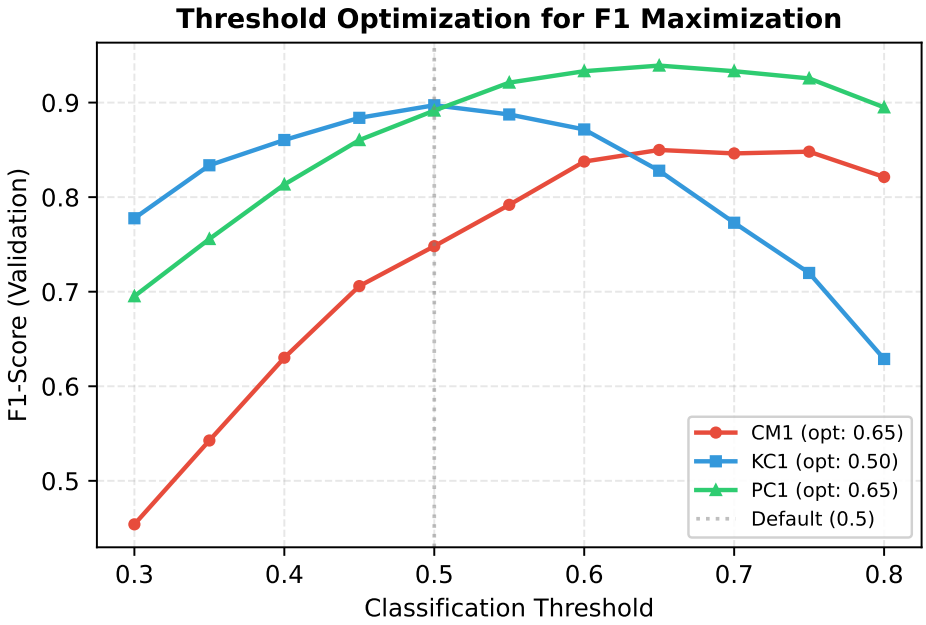


Figure 4.1. Threshold calibration analysis for different datasets.

4.9 SHAP Explainability Analysis

SHAP explainability analysis is integrated to improve interpretability of deep learning predictions.

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i \tag{4.11}$$

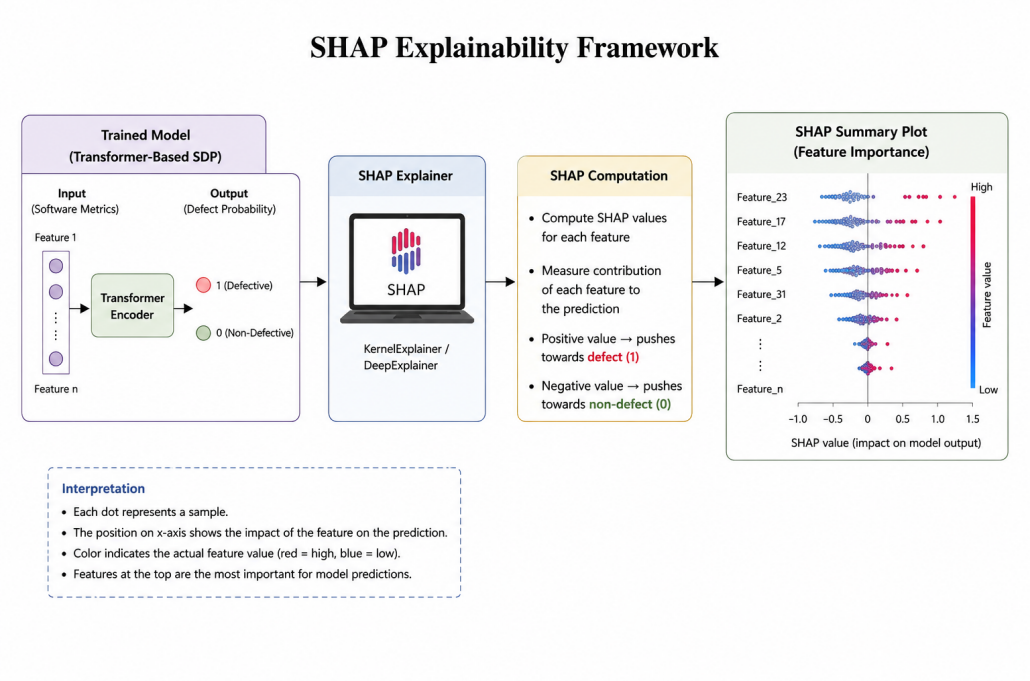


Figure 4.2. SHAP-based explainability workflow.

CHAPTER 5

EXPERIMENT RESULTS AND DISCUSSION

5.1 Experiment Setting

Frameworks under consideration are built using Python, Scikit-learn, TensorFlow, Keras, LightGBM, XGBoost, Optuna, and SHAP frameworks.

Experiments are performed on the CM1, KC1, and PC1 data collections taken from the NASA PROMISE repository.

Table 5.1. Configuration settings for the experiment

Component	Configuration
Programming language	Python 3.x
Machine learning library	Scikit-learn
Deep learning framework	TensorFlow/Keras
Optimization framework	Optuna
Hardware	Intel Core i7, 32 GB RAM

5.2 Performance Measures

Measures that were used during evaluation include accuracy, precision, recall, F1-score, and ROC-AUC.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5.4}$$

5.3 Optimization-Based Framework Results

Optimization-based ensemble framework showed strong performance when applied to benchmark datasets.

Table 5.2. Performance of optimization-based ensemble framework

Dataset	Accuracy	Precision	Recall	F1-score
CM1	0.906	0.910	0.900	0.905
KC1	0.924	0.937	0.910	0.923
PC1	0.964	0.961	0.966	0.964

5.4 Transformer-Based Framework Results

Transformer-based framework showed the ability to detect complex relationships between software metrics.

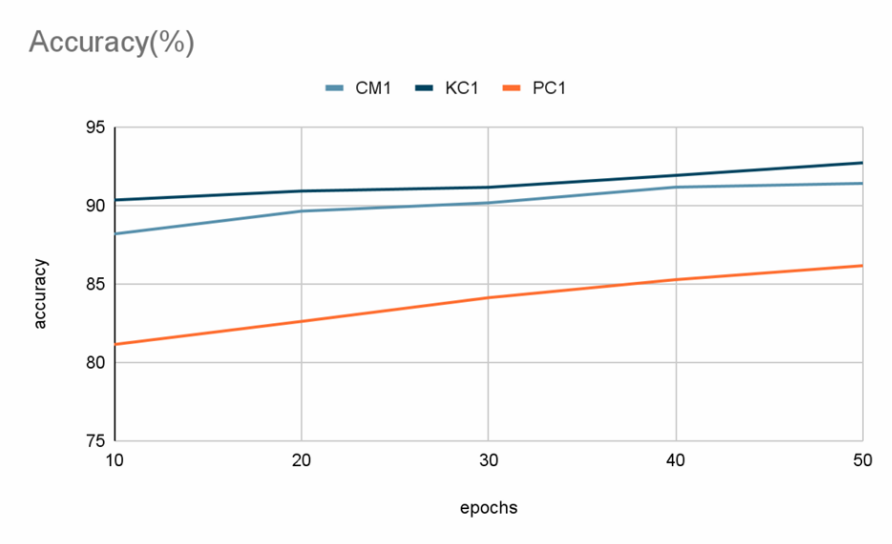


Figure 5.1. Comparative performance of transformer-based framework

The graph illustrates comparative performance of the proposed framework.

5.5 ROC Curve Analysis

Analysis of ROC curves shows considerable discriminative capabilities of proposed approaches.

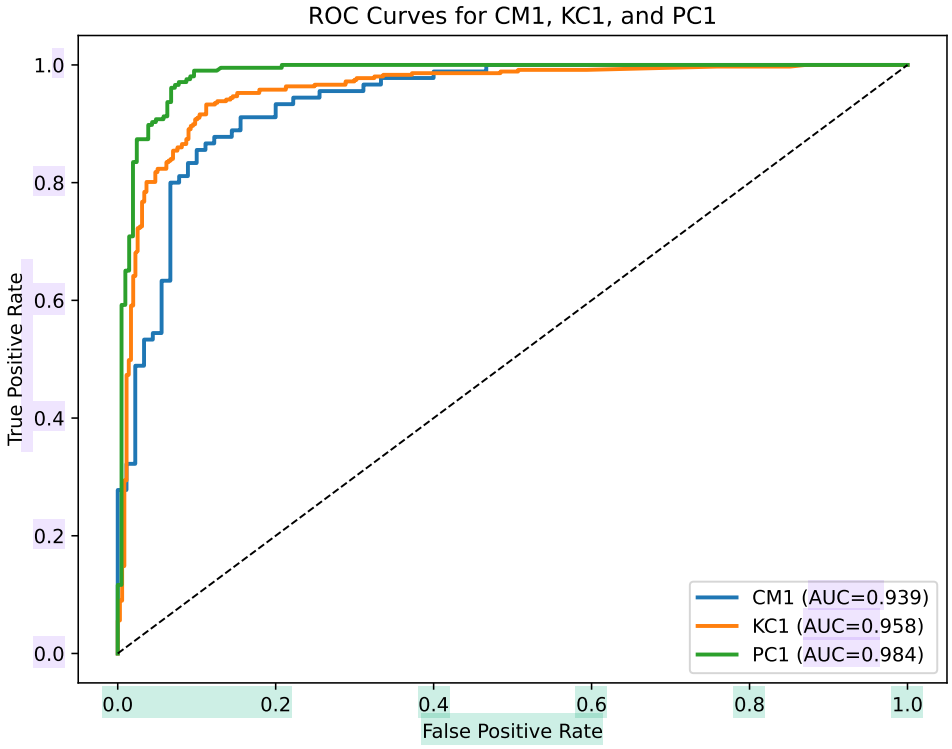


Figure 5.2. ROC curve analysis for optimization-based framework

5.6 Ablation Study

In order to understand the contribution of the specific components, an ablation study was carried out.

Experimental results reveal the following:

- SMOTE-ENN provides a substantial boost to recall score.
- Binary PSO reduces number of features.
- Threshold calibration helps to improve F1-score performance.
- Transformer encoders boost representation learning capabilities.

5.7 Feature Reduction Analysis

Feature reduction analysis demonstrates that Binary PSO effectively removes redundant software metrics.

5.8 Threshold Analysis

Threshold calibration analysis demonstrates that optimal threshold values differ across datasets.

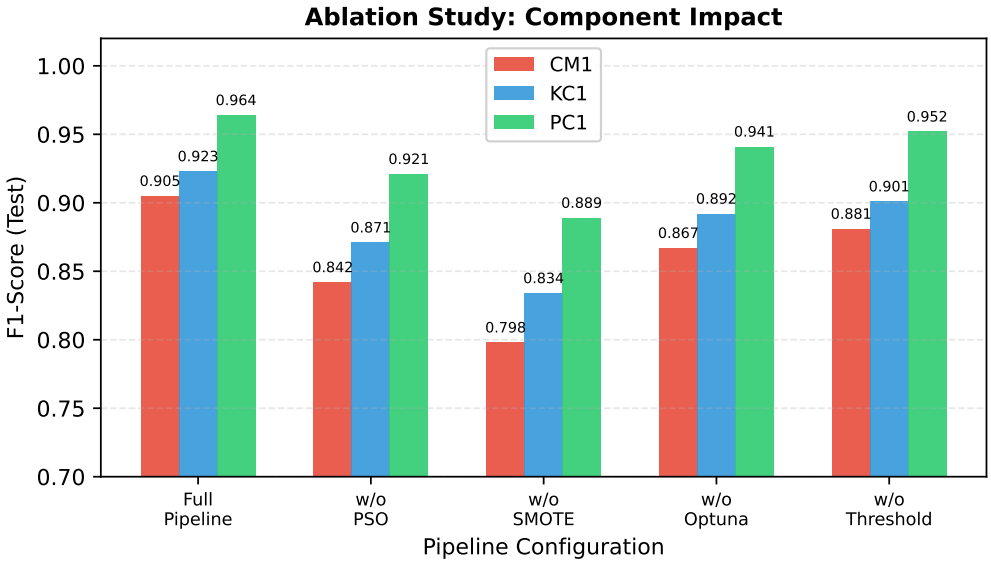


Figure 5.3. Ablation study showing impact of components on F1-score

Table 5.3. Feature Selection Efficiency

Dataset	Original Features	Selected Features	Reduction (%)
CM1	38	7	81.6%
KC1	22	14	36.4%
PC1	21	11	47.6%

5.9 Comparative Analysis

The optimization-based framework provides lower computational complexity and efficient feature optimization.

The Transformer-based framework provides stronger representation learning and improved interpretability using SHAP explainability analysis.

Table 5.4. Comparative Analysis of Proposed Frameworks

Criterion	Optimization Framework	Transformer Framework
Feature Selection	Binary PSO	Automatic Representation Learning
Computational Complexity	Lower	Higher
Interpretability	Moderate	High using SHAP
Training Time	Faster	Slower
Minority-Class Learning	Strong	Very Strong

5.10 Key Findings and Limitations

The major findings are summarized below:

1. Binary PSO effectively reduces feature dimensionality.

2. SMOTE-ENN significantly improves minority-class prediction performance.
3. Threshold calibration improves F1-score performance.
4. Transformer encoder architecture improves representation learning capability.
5. SHAP improves model interpretability.

The limitations of the proposed work include:

- evaluation limited to NASA PROMISE datasets,
- higher computational complexity of Transformer models,
- limited cross-project evaluation.

CHAPTER 6

CONCLUSION

6.1 Conclusion

Early detection of software modules prone to defects is a crucial activity in enhancing software quality, reliability and maintenance efficiency, a task that can be addressed by Software Defect Prediction (SDP). But, difficulties like class imbalance, high dimensional feature space, redundant software metrics, lack of interpretability and inefficient representation learning still exist in the effectiveness of the existing SDP approaches.

This thesis introduced a thorough framework of software defect prediction that combines optimization-based ensemble learning with explainable deep learning methods using Transformers. The proposed work integrated feature selection, imbalance handling, hyperparameter optimisation, threshold calibration, ensemble learning, deep learning and explainable artificial intelligence in the process of making predictions, all under a single roof.

The first proposed framework applied Binary Particle Swarm Optimization (PSO) algorithm for feature selection, SMOTE-ENN algorithm for handling imbalanced data and Optuna algorithm for hyperparameter optimisation to the software defect prediction problem, using a LightGBM-XGBoost soft voting ensemble classifier. Experimental results showed that the framework successfully lowered the dimensionality of features and enhanced minority-class prediction ability and the overall classification reliability.

The second proposed framework combined convolutional feature extraction, Transformer encoder architecture, focal loss and explainability analysis using SHAP. The Transformer-based framework exhibited strong learning capability of complex relationship between software metrics with the help of self-attention mechanism. SHAP explainability analysis further enhanced transparency by selecting software metrics that are important for making the defect prediction decisions.

For both proposed frameworks, good prediction performance was shown by experimental evaluation on NASA PROMISE datasets in terms of Accuracy, Precision, Recall, F1-score and ROC-AUC. Comparative analysis showed that the optimization-based framework has lower computational complexity and efficient feature optimization, and the Transformer-based framework has better representation learning capability and inter-

pretability.

These are the proposed approaches contributing to the development of a reliable, interpretable and scalable software defect prediction system that can be adapted to the software engineering environment.

6.2 Future Directions

There are several interesting research avenues that can further develop software defect prediction systems.

The proposed frameworks can be evaluated using large-scale industrial software repositories and cross-project defect prediction datasets in the future to enhance the generalisation across different software systems.

Transformer architectures with lower computational complexity and lower weights may enhance the applicability of deep learning-based software defect prediction frameworks in resource-limited settings.

Other studies may be conducted:

- cross-project software defect prediction using transfer learning, and
- software representation learning using graph neural network, and
- software defect prediction using source code and commit history, and
- adaptive threshold optimization strategies,
- real-time Just-In-Time software defect prediction systems.

Other explainable AI methods could also yield greater transparency, explainability, and trust in the automated software defect prediction systems developed by developers.

In the future, software engineering environments will benefit from intelligent software quality assurance systems, which will be even more effective thanks to the integration of advanced deep learning architectures and optimization-based ensemble learning methods.

BIBLIOGRAPHY

[1] "Promise software engineering repository," available at: http://promise.site.uottawa.ca/SERepository.

[2] S. Pal and A. Sillitti, "Cross-project defect prediction: A literature review," *IEEE Access*, vol. 10, pp. 118 697–118 717, 2022.

[3] A. Abdu, Z. Zhai, H. Abdo, R. Algabri, M. Al-Masni, M. Muhammad, and Y. Gu, "Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model," *Scientific Reports*, vol. 14, no. 1, p. 14771, 2024.

[4] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[8] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[9] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[10] Q. Zhang and B. Wu, "Software defect prediction via transformer," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1. IEEE, 2020, pp. 874–879.

[11] A. Mishra and A. Sharma, "Deep learning based continuous integration and continuous delivery software defect prediction with effective optimization strategy," *Knowledge-Based Systems*, vol. 296, p. 111835, 2024.

[12] Y. Al-Smadi, M. Eshtay, A. Al-Qerem, S. Nashwan, O. Ouda, and A. Abd El-Aziz, "Reliable prediction of software defects using shapley interpretable machine learning models," *Egyptian Informatics Journal*, vol. 24, no. 3, p. 100386, 2023.

[13] M. Begum, M. Shuvo, I. Ashraf, A. Al Mamun, J. Uddin, and M. Samad, "Software defects identification: Results using machine learning and explainable artificial intelligence techniques," *IEEE Access*, vol. 11, pp. 132 750–132 765, 2023.

[14] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[15] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[16] R. Malhotra and A. Bansal, "Use of support vector machine to check whether process metrics are as good as static code metrics," in *Topical Drifts in Intelligent Computing*. Springer Nature Singapore, 2022, pp. 35–42.

[17] A. Jude and J. Uddin, "Explainable software defects classification using smote and machine learning," *Annals of Emerging Technologies in Computing (AETiC)*, vol. 8, no. 1, pp. 36–49, 2024.

[18] S. Mohapatra, E. Ju, J. Lee, and D. Ryu, "Software defect prediction based on saint," *The Transactions of the Korea Information Processing Society*, vol. 13, no. 5, pp. 236–242, 2024.

[19] R. Malhotra and J. Patidar, "Enhanced software change proneness prediction in android applications using balanced data techniques and advanced language models," *Discover Computing*, vol. 28, no. 1, p. 51, 2025.

[20] G. Santos, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, "Predicting software defects with explainable machine learning," in *Proceedings of the XIX Brazilian Symposium on Software Quality*, 2020, pp. 1–10.