ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

Enhancing Cross-Project Defect Prediction Using Stacking Ensemble and Hybrid Sampling

Aliyu Bashir Nuhu[1]; Dr. Yusuf Salisu Ibrahim[1]

[1]Department of Software Engineering, Faculty of Computing, Nile University of Nigeria

Publication Date: 2026/05/13

Abstract: Software Defect Prediction known as SDP improves software quality by identifying defect-prone modules early in development, however, Cross-Project Defect Prediction (CPDP) remains challenging due to data heterogeneity across projects and severe class imbalance in defect datasets. Conventional machine learning models fail in effective generalization of new projects and demonstrate poor minority class detection in projects. This study aimed to develop and evaluate a Hybrid Ensemble Deep Learning framework to improve Cross Project Defect Prediction and performance under heterogeneous and imbalanced conditions. The framework combines Random Forest and XGBoost as base learners in a stacking generalization architecture, after which a Deep Neural Network serves as the meta-classifier. To address class imbalance and boundary noise, a SMOTE-Tomek hybrid sampling technique was joined into the preprocessing pipeline. The model was evaluated using a Leave-One-Project-Out (LOPO) validation approach on 5 different PROMISE datasets (CM1, JM1, KC1, MW1, and PC1). The results gotten demonstrated strong cross-project generalization ability, also achieving an average Accuracy of 88.63%, Recall of 83.72%, F1-score of 0.77, and AUC of 0.76. The framework maintained high defect detection capability without unnecessary false alarms. The study concludes that combining stacking ensembles, deep meta-learning, and hybrid sampling provides a pragmatic and computationally doable solution for enhancing Cross-Project Defect Prediction abilities.

Keywords: *Hybrid Ensemble Deep Learning; Cross-Project Defect Prediction; Federated Meta Learning; Software Defect Prediction; Ensemble Learning; Synthetic Minority Over-Sampling Technique; Machine Learning.*

How to Cite: Aliyu Bashir Nuhu; Dr. Yusuf Salisu Ibrahim (2026) Enhancing Cross-Project Defect Prediction Using Stacking Ensemble and Hybrid Sampling. *International Journal of Innovative Science and Research Technology*, 11(5), 123-140. https://doi.org/10.38124/ijisrt/26may202

I. INTRODUCTION

➢ *Background of the Study*

Software testing as a component of modern day software engineering is among the main methods of measuring whether applications are performing up to the required expectations, whether or not they are achieving successes and a level of consistency despite varying circumstances (Akhtar, 2025).

Advanced methods of testing are not only about identifying errors; they are about making user experience better by enhancing efficiency and speed, and the management and testing of the necessary security standards, such as GDPR (Savvycom, n.d.). The growing difficulty of modern software systems suggests that quality assurance has moved beyond testing of mere features, to include much more of non-functional features. This change is a factor to the increased need of high-level risk mitigation, this is even more so in a rapid development and continuous integration environment (Savvycom, n.d.). Therefore, proactive and predictive software risk management has become an important business need that directly touches on customer satisfaction and decreases the high financial cost of fixing the post-release defects (Savvycom, n.d.).

Nevertheless, existing and manual software testing methods are becoming less effective the more when challenged by the speed and complexity of agile development life cycles, especially in Continuous Integration/Continuous Delivery (CI/CD) pipelines (Kesavan, 2025). Manual testing is also susceptible to human error when reviewing and detecting test results, and it does not scale to the size of large and continuously changing codebases, resulting in high maintenance costs of test cases (Arai, 2025). Although automation aids in curtailing repetitive work, the challenge in automating quality assurance is in the fact that it is difficult to recreate the human intuition in less straightforward cases and the cost and experience necessary to establish and manage dynamically changing test scripts (Medium, n.d.). This failure of traditional approaches to manage the twofold threats of complexity and speed compels a paradigm shift towards smart and data-driven approaches that may automate resource-consuming and predictive processes (Jayasekera, 2025).

Applying Machine Learning (ML) to software testing is a advent that may help to better quality assurance procedures in terms of efficiency, accuracy, and scope by means of intelligent automation and prediction (Bakar, 2024). According to previous testing results, software metrics, and established

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

defect patterns, ML algorithms have the potential to shift the software quality assurance activity, which has only been reactive, to a proactive risk management discipline (Kesavan, 2025). The development teams can use specific high-impact ML applications, including Software Defect Prediction (SDP) and Test Suite Prioritization (TSP), to devote scarce inspection resources to those code fragments most likely to harbor faults, thereby greatly boosting the rate at which continuous integration can be achieved and lowering expenses (Wojcicki & Dabrowski, 2018). This study aims to enhance prediction ability, specifically defect prediction which is arguably the most important to use where it is consequential to anticipate the resources early on and better software reliability (Albattah and Alzahrani, 2024).

➢ *Problem Statement*

The application of Machine Learning (ML) in Software Defect Prediction (SDP) is limited by several critical challenges, particularly in Cross-Project Defect Prediction (CPDP). Models trained on a single project often fail to generalize effectively to new projects due to differences in data distribution. This issue is compounded by severe class imbalance in defect datasets, where non-defective modules significantly outnumber defective ones. As a result, many traditional models achieve high overall accuracy but perform poorly in detecting defects, which is the most critical objective in real-world software engineering.

Although advanced approaches such as federated meta-learning improve generalization, they are often computationally expensive and impractical for widespread use. On the other hand, simpler models lack the ability to capture complex patterns across heterogeneous datasets, leading to inconsistent performance. Furthermore, existing studies tend to emphasize model complexity while overlooking effective data preprocessing techniques, particularly hybrid sampling methods for handling class imbalance. This creates a gap for a practical, efficient, and high-performing solution that can address both data heterogeneity and class imbalance in CPDP.

Despite advances in Machine Learning, existing models still struggle with data geterogeneity and severe class inbalance, leading to unreliable predictions across projects. Therefore there is a need for a robust hybrid learning framework that improves generalization and prediction performance across heterogenous software projects.

➢ *Research Aim and Objectives*

The overarching aim of this research is to advance the application of Machine Learning in software testing by developing a robust, generalizable framework for Cross-Project Defect Prediction (CPDP) capable of mitigating the persistent challenges of data heterogeneity and extreme class imbalance.

- *To Achieve this Aim, the Specific Objectives of the Research are:*

✓ To formulate and develop a Hybrid Ensemble Deep Learning (HEDL) system utilizing a stacking generalization architecture, which integrates the predictive capabilities of robust base classifiers (Random Forest and XGBoost) with a Deep Neural Network (DNN) meta-learner to effectively capture complex, non-linear defect patterns across software projects.

✓ To design and implement an advanced data preprocessing pipeline tailored for the HEDL framework, incorporating Independent Log-Standardization and a hybrid sampling strategy (SMOTE-Tomek Links) to systematically resolve the effects of extreme class imbalance and distributional heterogeneity inherent in publicly available CPDP datasets.

✓ To empirically evaluate the performance and generalizability of the proposed HEDL framework against traditional baseline models using a rigorous Leave-One-Project-Out (LOPO) cross-validation strategy, measuring predictive efficacy through standard classification metrics including Accuracy, Precision, Recall, F1-Score, Area Under the ROC Curve (AUC), and Geometric Mean (G-Mean).

➢ *Scope and Significance*

The study is important both to academic research and to industrial software engineering. In academia, the framework developed will provide a forward-looking research avenue by providing a Hybrid Ensemble Deep Learning (HEDL) framework essentially for providing a solution to the compounded problem of data heterogeneity and class imbalance in Cross-Project Defect Prediction (CPDP), thus satisfying a known gap in the current research on the practical, high-performance CPDP solution. The rigorously tested Hybrid Ensemble Deep Learning (HEDL) framework will serve as a reliable metric to predict the effect of ensemble stacking and specialized data sampling methods within the model.

This study is limited to the supervised machine learning methods of Software Defect Prediction (SDP) as applied to Cross-Project Defect Prediction (CPDP). The paper that will be studied is limited to the use of publicly available, static code metrics and change metrics derived and obtained through the established repositories, including PROMISE and AEEEM. The research itself is restricted to using and analyzing the suggested Hybrid Ensemble Deep Learning (HEDL) framework, and in particular, the integration and effectiveness of hybrid sampling techniques (SMOTE-Tomek links) to the effect of mitigation of the impact of class imbalance. Not covered are applications of ML to dynamic program analysis, exploratory analysis to produce general test cases (e.g. using Large Language Models or LLMs), applications of explainable AI (XAI) model interpretability, and application of proprietary or highly specialized industrial data to model training.

II. REVIEW OF RELATED LITERATURE

This section gives a review of the existing academic literature related to Machine Learning applications in software testing, with a peculiar focus on Software Defect Prediction. The review is structured to analyze the evolution of defect prediction from conventional Within-Project methods of prediction to the more advanced methods i.e Cross-Project Defect Prediction (CPDP). It also profoundly reviews the two challenges of data heterogeneity and class imbalance,

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

evaluating modern methodologies such as Federated Learning, Ensemble techniques, and Deep Learning architectures. Lastly, the chapter establishes the Theoretical Framework corroborating the proposed Hybrid Ensemble Deep Learning (HEDL) system.

➢ *Overview of Software Defect Prediction*

Software Defect Prediction (SDP) has become a crucial subdivision in software engineering, aiming to identify possible defect vulnerabilities even before testing commences. By capitalizing on existing data and software metrics, SDP models gives room for quality assurance teams to apportion limited resources accordingly.

- *The Shift to Intelligent Automation*

Contemporary literature suggests and iterates a shift from manual testing towards intelligent automation. Akhtar (2025) suggests that software testing has ceased to be about error detection but about ensuring quality and consistency in robust environments. This view is supported by Bakar (2024), who assessed the implementation of ML in automated testing and established that ML driven application crucially reduces manual intervention while also improving defect detection rates. The importance of this shift is propelled by the rate at which modern Continuous Integration (CI) pipelines is scaling, where traditional methods struggle to scale (Kesavan, 2025).

- *Within-Project vs. Cross-Project Prediction*

Before now, Software Defect Prediction research was around Within-Project Defect Prediction(WPDP)mainly, in which models are trained and tested on data from the same project. Nevertheless, this technique suffers from the "cold-start" problem, whereby new projects don't have the historical data to train models. This development brought about the advent of Cross- Project Defect Prediction (CPDP), where data from established projects(source) is used to predict defects in new projects (target). Even though CPDP brings about a solution to data scarcity, it also brings about enormous problems related to data distribution shifts, which is amongst the focus of modern research (Saeed & Saleem, 2023).

➢ *The Challenge of Heterogeneity in CPDP*

A major bottleneck in Cross-Project Defect Prediction is the heterogeneity between source and target projects. Software metrics such as Cyclomatic Complexity, Lines of Code, frequently follow different statistical distributions across several development teams and domains, resulting in very poor model generalization.

- *Federated and Transfer Learning Approaches*

To tackle the issue of heterogeneity, modern research have reviewed complex architectures. Chen et al. (2024) proposed an Efficient Communication-based Federated Meta-Learning (ECFML) framework. Using MobileViT as a meta-learner and integrating privacy-preserving mechanisms, they attained significant advancements in AUC (0.7208) on the AEEEM dataset. Similarly, Wang et al. (2024) introduced "FedDPI," a federated learning approach using differential perception to manage cross-project data. Even though these methods show high performance, they are characterized by extreme architectural complexity. Wang et al. (2024) noted that such models require highly structured data allocation and complex optimization stages, which may limit their functionality in resource-constrained environments.

- *Clustering and Optimization*

Unconventional methods aim to normalize heterogeneity through clustering. Jindal et al. (2021) invented a hybrid model comprising of an Ensemble Learning with Genetic Algorithms (GA) and a k-means clustering technique. Albeit they achieved a moderate F1-score of 0.666 on PROMISE data, the reliance on k-means clustering was defined as a limitation, because clustering usually fails to detect the complex, non-linear defect patterns in heterogeneous software projects.

➢ *Class Imbalance in Software Engineering*

Another challenge as pointed out in the literature is class imbalance. In defect datasets(like PROMISE or NASA), non-defective modules greatly surpass defective modules. Standard classifiers trained on such data tend to have bias towards the majority class, which results in high accuracy but then fail to detect real defects.

- *Impact on Predictive Performance*

Nassif et al. (2023) did a comparative research on Learning to Rank (LTR) models, they observed while "bug count" was a stable target variable, the presence of class imbalance adversely affected the model's performance unless expressly addressed. Their research findings suggest that standard optimization methods are usually not sufficient without dedicated imbalance learning techniques.

- *Resampling Techniques*

In an attempt to curtail this, researchers employ resampling techniques. Mehta et al. (2025) used a Bi-LSTM network integrated with Random Oversampling and SMOTE (Synthetic Minority Oversampling Technique). They achieved satisfactory results, showing a 43% improvement in F-measure as against imbalanced baselines. Bennin et al. (2022) warned that while simple resampling betters minority class detection, it can introduce noise. This is the highlight of the gap discovered by this research: the need for hybrid sampling methods e.g SMOTE-Tomek that oversample the minority class and as well clean the decision boundaries.

➢ *Ensemble and Deep Learning Approaches*

The literature shows an obvious drift to the direction of Ensemble Learning and Deep Learning to manage non-linear complexities of software code.

- *Ensemble Learning*

The combination of multiple classifiers, as in Ensemble methods, to reduce bias as well as variance became widely adopted. Rahman and Eisty (2025) showed how important the use of ensemble algorithms for auto-test case generation and selecting test cases with high diversity to better coverage. while in the area of predicting defects, Kumar and Saxena (2024) compared various hybrid frameworks, understanding that a stacking ensemble (consisting GaussianNB, KNeighbors, and Neural Networks) did better by far than single classifiers when it came to Recall, AUC and ROC. This

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

is a crucial breakthrough because it encourages the "stacking" technique adopted in this research.

- *Deep Learning and Feature Representation*

No doubt, Deep Learning provides enhanced capabilities for feature representation. Qu et al. (2025) created a technique that employs deformable attention mechanisms to pull semantic features from Abstract Syntax Trees (ASTs). Their integration of semantic and network features exceeded benchmarks by as much as 41%. In a similar vein, Harsh et al. (2025) verified through a comparative study that advanced feature extraction methods by far surpass current leading baselines.

- *Large Language Models (LLMs)*

Latest studies are also reviewing Generative AI. Chen et al. (2024) showcased "ChatUniTest," a framework seen to use large language models for the purpose of generating unit tests. Although this shows potential, Nyaga (2025) illustrates that these models face challenges related to low interpretability and high cost of computation, which therefore makes existing supervised machine learning models more realistic in terms of speedy implementation in defect prediction in the industry.

➢ *Theoretical Framework*

This study is based on three core concepts: Stacking Generalization, Hybrid Sampling and Deep Neural Networks.

- *Stacking Generalization Theory*

An ensemble learning technique called stacking, or stacked generalization, with the aim to mitigate generalization error for some model sets. Stacking uses a two-tier architecture, the suggested Hybrid Ensemble Deep Learning (HEDL) method, contrary to Bagging, which averages predictions, or "Boosting," which corrects errors in a sequential manner.

✓ *Level - 0 (The Base Learners):*

This is a set of different classifiers (e.g., Random Forest, XGBoost) $C_1, C_2, ..., C_n$, each predicts the target variable seperately.

✓ *Level - 1 (The Meta-Classifier):*

This is a meta learner that will take the predictions from the base learners or Level-0 models as input features to learn the optimal combination logic. Mathematically, if x is the input vector, the meta classifier M learns a function F such that:

$$\hat{y} = F\,(C_1(x)\,, C_2(x)\,, ..., C_n(x))$$

Eqn 1 Stacking Generalization Equation

Where:

$\hat{y}$ = Final predicted output (class label or probability of a module being defective)

$F(\cdot)$ = Meta-classifier function (Deep Neural Network in this study)

$C_1, C_2, ..., C_n$ = Base learners (e.g., Random Forest, XGBoost)

$C_i(x)$ = Prediction output of the ith base learner for input sample x

x = Input feature vector representing a software module

This enables the system to know which of the base classifiers is dependable for certain kinds of data points, efficiently handling the heterogeneity present in CPDP (Kumar & Saxena, 2024).

- *Deep Neural Networks (DNN) as Meta-Learners*

Albeit Logistic Regression as well as other simple learning algorithms are frequently chosen as meta-learners, in this study, a Deep Neural Network (DNN) is used because of their ability to model advanced, non-linear relationships and also are universal function approximators. The DNN captures advanced interactions that linear meta-learners may not capture in the setting of HEDL by mapping the probabilistic outputs of the base learners to the final class label.

- *SMOTE-Tomek Links (Hybrid Sampling Theory)*

To curtail the issue of class imbalance, this research will explore the framework of Hybrid Sampling, particularly the combination of SMOTE and Tomek Links.

✓ *SMOTE (Synthetic Minority Over-sampling Technique):*

This technique generates synthetic samples and not just simple duplication. Given a dataset, for a minority sample x_i, it finds a k-nearest neighbor x_{zi} and creates a new point x_{new} along the line segment joining them:

$$x_{new} = x_i + \lambda\,(x_{zi} + x_i)$$

Eqn 2 SMOTE Synthetic Sample Generation

✓ *Tomek Links:*

This is a technique in data cleaning by way of undersampling. A Tomek Link is said to exist if given two samples x_i, and x_j, from different classes are one other's nearest neighbor. Eliminating these pairs tidies the decision boundary, minimizing noise and overlap. By combining these (SMOTE-Tomek), the framework ensures both a balanced distribution by SMOTE and a clear decision boundary by Tomek Links, rightly addressing the limitations as raised by Bennin et al. (2022).

➢ *Review of Related Works*

There are a number of studies that have covered various issues related to the software defect prediction, machine learning application in software testing and ensemble learning. Table 1 lists a summary of the most significant related works published since 2021 and shows their methodology, findings, and limitations.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

Table 1 Summary of Related Works

S/N	Author (s) & Year	Methodology/Contributions	Results	Limitation / Gap
1	Chen et al. (2024)	Efficient Communication-based Federated Meta-Learning (ECFML) using MobileViT as a meta-learner for CPDP. Integrates Laplace mechanism for privacy.	Achieved average AUC of 0.7208 and G-mean of 0.5730 on AEEEM and Relink datasets, significantly outperforming baselines.	High architectural complexity and reliance on specialized communication protocols; may be resource-intensive for small organizations or non- specialized environments.
2	Rahman & Eisty 2025, May)	Ensemble Machine Learning Algorithms for Automatic Test Case Generation using Learning Based Testing (ELBT)	Selects test cases with high diversity in ensemble predictions to reduce the cost of testing a full SUT specification.	Focuses on black-box test case selection and may not optimize for internal code coverage or path analysis as effectively as specialized LLM methods.
3	Wang et al. (2024)	FedDPI: Federated Learning for CPDP with a two-stage collaborative optimization mechanism and differential perception	Average improvement in AUC (0.2783) and G-mean (0.2673) compared to other DL/FL algorithms, verifying practicality and effectiveness	Requireshighly structured data allocation and complex optimization stages; generalization relies heavily on advanced privacy protection strategies
4	Mehta et al. (2025)	Bi-LSTM network combined with random oversampling and SMOTE techniques for software fault prediction with imbalanced data	Improved average F- measure by 43% (random oversampling) and 41% (SMOTE) compared to original, imbalanced datasets	Focuses primarily on within-project fault data and does not fully address the combined challenge of cross-project heterogeneity
5	Qu et al. (2025)	Developed a method for efficient AST representation. Used deformable attention to extract individual features (semantic & network). Employed self-attention to effectively fuse these distinct features.	Fusion features outperformed single features. Cross-version average performance: ACC 0.7, F1 0.614, AUC 0.711. Cross-project average performance: ACC 0.687, F1 0.575, AUC 0.696. MFA showed up to 41% improvement over benchmarks.	Inherent complexity of software structures makes full feature utilization difficult. Small number of samples can reduce performance. Single feature type is insufficient for full characterization.
6	Jindal et al. (2021)	Hybrid model for CPDP combining ensemble learning (EL) and Genetic Algorithm (GA) optimization, utilizing k-means clustering on PROMISE data.	Achieved an F1 score of 0.666 after implementing k-means clustering for data normalization.	The F1-score suggests moderate performance; the approach relies on clustering which may fail to capture complex, non- linear defect patterns across heterogeneous projects.
7	Chen & Chen (2025)	Industrial Application Software Testing Framework (IAST) using task logic-based explanatory intelligence	Helps complete system testing for different functional IAS families, addressing needs for reliability and test efficiency with stringent explanatory intelligence	Focuses on system testing and requires task logic definition, which may limit applicability in early-stage code analysis and prediction tasks
8	Chen et al. (2024)	LLM-based automated unit test generation framework (ChatUniTest) with adaptive focal context mechanisms	Demonstrated potential for reliable tests with high pass rates and coverage, showcasing the role of LLMs in unit test generation	High-level test case generation remains under-explored, and LLMs often lack the ability to fully comprehend and cover specific program logic execution paths
9	Nyaga (2025)	Systematic literature review on ML in software engineering, including error detection and program maintenance	Identified a growing trend in Graph Neural Networks (GNNs) for code analysis, demonstrating ML's transformational potential in the SDLC	Highlights challenges such as low model interpretability, high computation costs, and difficulties in integrating ML into existing workflows
10	Bakar (2024)	Review of Machine Learning implementation in automated software testing, focusing on	ML significantly reduces manual intervention and improves defect detection	A general review that lacks specific empirical contributions regarding hybrid modeling or solution

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

		test case generation and defect prediction	rates, facilitating more reliable software delivery	comparisons for CPDP
11	Harsh et al. (2025)	Comparative Analysis of Machine Learning Models for Software Defect Prediction (SDP)	Demonstrated that advanced feature extraction and integration techniques significantly outperform state-of-the-art baseline models	Results focused primarily on accuracy and AUC; may overlook the crucial F1-score metric essential for imbalanced classification fidelity
12	Kumar et al. (2024)	Hybrid ML framework combining Deep Neural Networks with ensemble models (SVM, RF, XGBoost) for cross-project defect prediction	Hybrid Model 3 (KNeighbors, GaussianNB, SVC, Neural Network) surpassed others in recall, F1-score, accuracy, and ROC AUC	Model 3 uses a diverse but standard set of classifiers in its ensemble; the selection of base learners might not be optimally tuned for maximum CPDP feature resilience
13	Nassif et al. (2023)	Performed a comparative study of eight selected Learning to Rank (LTR) models in the predicting Software Defects (SDP) task. Two target variables to rank the LTR models were bug count and bug density. It was also observed how the imbalance learning and feature selection contributed to the LTR models. Measurement was done based on Fault Percentile Average.	Bug count generated better scores and more consistent scores than bug density as the ranking criteria. Bug density was positively affected by imbalance learning whereas bug count was negatively affected. Regarding bug density, feature selection did not exhibit significant improvement, but with respect to bug count, it did not wear off.	The conclusion is that there is a limitation on methods of enhancement: The results of feature selection and imbalance learning on LTR are not superior or better in this task, and these usual optimization methods may not be good at LTR in SDP.
14	Kand & Do (2024)	Case study utilizing embedded software data from Samsung Electronics telecommunication systems with nine novel features for SDP	F-measurement metric was enhanced from 0.58 to 0.63 (an 8.62% augmentation) upon integration of new features	The findings are specific to an embedded telecommunication system dataset, making generalization across diverse industrial sectors difficult without further research
15	Khan et al. (2025)	Review focusing on ML-based Test Case Prioritization (TSP) in Continuous Integration (CI) environments	Concluded that there is a growing need to explore other information sources (requirements, models) and address CI-specific problems like test case volatility and flaky tests	Did not propose a novel architecture; served as a review, highlighting the persistent challenges in applying ML to highly dynamic CI workflows

➢ *Research Gap*

The literature review makes it very clear that two most important gaps in research in machine learning application to software testing have come together in the two most serious problems; firstly, generalization in Cross-Project Defect Prediction (CPDP) models and, secondly, the systematic inability to directly incorporate highly effective data quality approaches as part of modern classification systems. Even though such advanced approaches as Federated Meta-Learning (FML) have better CPDP results (Chen, 2024), they are too complex and require a special framework to communicate effectively and protect privacy, which is not applicable in the context of available resources and academic studies (Nyaga, 2025). On the other hand, most popular methods listed under the category of ensemble approaches, though powerful, can frequently lack the ability to fully capture the extreme imbalance of classes and boundary noise that exist in PROMISE and other such datasets, leading to models that, although accurate overall, have low recall and F1-scores on the important minority class(Saeed & Saleem, 2023). This creates an empirical requirement in a single, stacked classification model that has a trade-off between predictive accuracy and structural simplicity, and is explicitly intended to address both heterogeneity and imbalance among classes at the same time.

The research is therefore motivated by the necessity of coming up with a high-quality CPDP solution which is architecturally feasible and universal. The suggested Hybrid Ensemble Deep Learning (HEDL) framework is explicitly designed to solve the combined problem by combining the intrinsic resiliency of stacking generalization (reducing project heterogeneity) with an optimized, non-linear mapping task offered by a DNN meta-classifier (Kumar & Ssaxean, 2024). The distinguishing step which is critical is the programmatic addition of a hybrid sampling method (e.g., SMOTE-Tomek links) to the specified preprocessing system, which is an effective technique to address the bias of a single

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

class and boundary noise a factor often regarded as secondary when studying complex architectures(Mehta, 2025). This study has the capability to offer a practical and methodologically viable replacement for complex SOTA models by applying the coherent HEDL framework with strict adherence to predefined baselines. This will provide substantial and meausrable benefits to defect prediction and resource allocation in practice, particularly within the software engineering process (Li et al., 2024).

The Hybrid Ensemble Deep Learning (HEDL) framework research as presented, addresses the notified limitations by offering a common ground solution i.e utilizing the efficiency of Stacking Generalization to manage the issue of heterogeneity and the accuracy of SMOTE-Tomek Links to guarantee data quality, all within an architecture that is more accessible than solutions relative to Federated Learning.

III. METHODOLOGY

This section shows in great detail the framework used to achieve the objectives of the research methodologically. the chapter elaborates on the development of the hybrid Ensemble Deep Learning Framework. The chapter also elaborates on the dataset selection, research design, the model's architectural specifications, the preprocessing piprline,as well as the procedures used in the validation of the performance of the model in CPDP scenarios.evaluation parameters, which were used as a verification of the developed system.

➢ *Research Design*

This study uses a Quantitative Experimental Research Design approach. This approach is selected to factually certify that the hypothesis that a stacking ensemble framework combined with hybrid sampling can statistically do better than traditional single-classifier models in heterogeneous defect prediction tasks.

- *The Experimental Design Goes by a Cross-Project Validation Sequence:*

✓ Source Domain Selection: Training the model on data from familiar software projects.
✓ Target Domain Evaluation: Testing the trained model on a completely new and independent project.
✓ Comparative Analysis: Evaluating the HEDL framework as against single classifiers such as Random Forest, XGBoost and standard Deep Learning models without any hybrid sampling.

➢ *Dataset Description*

This research makes use of datasets from known and reputable repositories in the software defect prediction community, namely, PROMISE and AEEEM. both datasets are representative of real world software systems being developed in C/C++ and Java.

Table 2 Gives a summary of the projects selected for this research. The model is tested against different levels of data heterogeneity and defect rates as contained in the projects.

Table 2 Summary of Selected Datasets (PROMISE Repository)

Project	Description	Language	Modules	Defect Rate (%)
CM1	NASA spacecraft instrument written in C	C	~498	~9.8%
JM1	Real-time ground system	C	~10,885	~19.3%
KC1	Storage management system	C++	~2,109	~15.4%
PC1	Earth orbiting satellite flight software	C	~1,109	~6.9%
MW1	Zero-gravity experiment framework	C	~403	~7.6%

- *The Datasets Entail Static Code Metrics that Measure the Structure and How Complex the Software Code is. Some of the Key Features Include:*

✓ McCabe Metrics: Cyclomatic Complexity, Essential Complexity.
✓ Halstead Metrics: Program vocabulary, Program length, Volume, difficulty, Effort.
✓ Loc Metrics: Lines of Code, Comment lines, Blank lines.
✓ Change Metrics: Entropy of source code changes, churn, and developer count.

➢ *The Proposed HEDL Framework Architecture*

The main contribution of this study is the development of the Hybrid Ensemble Deep Learning (HEDL) framework. It uses a Stacking Generalization approach, partitioned into two different levels.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

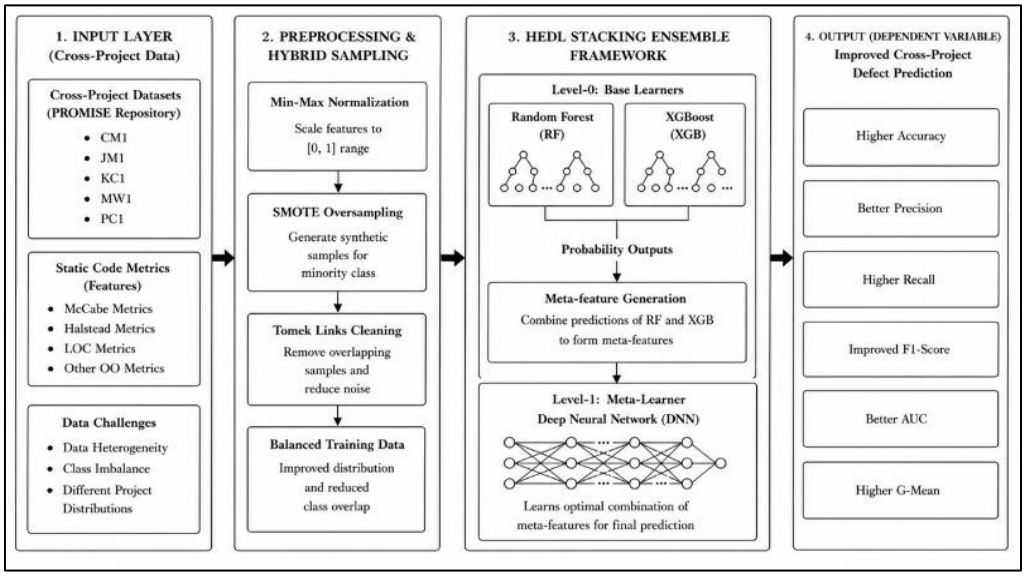

Fig 1 HEDL Conceptual Framework

- *Level-0: Base Learners*

The first layer or Level 0 consists of strong, diverse classifiers that are meant for extraction of initial patterns from the raw features.

✓ *Random Forest (RF):*

This is a bagging ensemble that constructs a numeorus decision trees at training time. It is chosen for its efficiency against noise and its capability of handling high-dimensional data without overfitting.

✓ *Extreme Gradient Boosting (XGBoost):*

This is a boosting algorithm that builds sequential trees, with each new tree managing errors made by the previous tree. It is chosen for its high execution speed rate and how well it performs in structured data classification.

✓ *Output:*

The probability scores generated by Random Forest and XG-Boost for the "Defective" class are extracted and linked together to form a new "meta-feature" vector.

- *Level-1: Meta-Classifier (Deep Neural Network)*

The second layer is a Deep Neural Network that takes the meta-features from Level-0 i.e the Base Learners as input.

✓ *Role:*

The DNN acts as a non-linear aggregator. Unlike simple averaging, the DNN learns advanced relationships between the base learners' confidence scores, effectively deciding which base learner to trust for specific types of data instances.

✓ *Structure:*

- Input Layer: Accepts the vector of probability scores.
- Hidden Layers: Two dense layers with Rectified Linear Unit (ReLU) activation functions to introduce non-linearity.
- Output Layer: A single neuron with a Sigmoid activation function to output the final probability of a module being defective (0 to 1).

- *HEDL System Flowchart*

The flowchart below depicts the end-to-end processing pipeline of the proposed methodology, from data acquisition to final part i.e defect prediction.

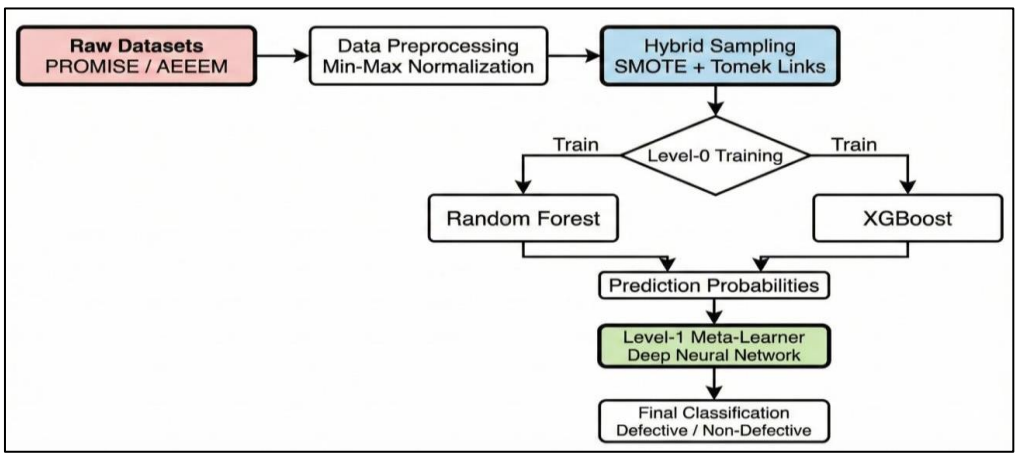

Fig 2 HEDL System Flowchart

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

➢ *Data Pre Processing and Imbalance handling*

With low defect rates as seen in Table 3.1 PC1 has just about 6.9% defects, standard training would result in a biased model. This research suggests a vigorous pipeline to curtail this.

- *Feature Scaling*

Given that metric values may differ great largely, for instance, Line of code (LOC) can be 1000 while Cyclomatic Complexity is 5. All input variables are scaled using Min-Max Scaling to ensure that values range between [0, 1]. This is done to stop the DNN and SMOTE distance calculations from bias in favor of variables with larger values.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Eqn 3 Min-Max Normalization Equation

Where:

X = Original value of the feature
X_{min} = Minimum value of the feature in the dataset
X_{max} = Maximum value of the feature in the dataset
X_{new} = Normalized value of the feature scaled to the range [0, 1]

- *Hybrid Sampling Strategy (SMOTE-Tomek)*

The approach method used in this study is a mixture of oversampling and undersampling techniques to balance the training data before it is passed to the base level classifiers.

✓ *Oversampling (SMOTE):*

The Synthetic Minority Over-sampling Technique (SMOTE) is used on the minority class (defects). It generates new artificial defects of the minority class by inserting them between existing defects of the minority class.

✓ *Undersampling (Tomek Links):*

When oversampling is completed, it is likely that the training set may have "Tomek Links," which are pairs of instances belonging to seperate datasets that are nearest neighbors to each other. These neighbouring pairs are often considered noise or instances near the boundary of a decision class. The Tomek Links algorithm eliminates the majority class instance from these pairs.
Result: The result is a balanced training set with noise-free decision boundaries.

➢ *Model Training and Validation Procedure*

This section details the training strategy employed to ensure the HEDL framework generalizes well across different software projects.

- *Training Strategy: Leave-One-Project-Out(LOPO)*

To rigorously test the Cross-Project Defect Prediction (CPDP) capabilities, a Leave-One-Project- Out validation scheme is used:

✓ Given a number of sets N (e.g., CM1, JM1, KC1, PC1, MW1), the model is trained on N-1 sets (Source Domain).
✓ The non-utilized set is used then as the test set (Target Domain).
✓ The same process is repeated N times so that every project serves as the target set once.
✓ This simulates the real-world scenario where a company has historical data from past projects but needs predictions for a brand-new project.

- *Hyperparameter Tuning (level-0)*

Base learners will be trained using a Grid Search with 5-layered cross-validation over the training set to pick the most suitable hyperparameter configuration:

✓ Random Forest: Hyperparameter tuning of *n_estimators* (50, 100, 200) and *max_depth* (10, 20, None).
✓ XGBoost: Hyperparameter tuning of *learning_ rate* (0.01, 0.1), *max_ depth* (3, 5, 7), and *n_ estimators.*

- *DNN Meta-Learner Optimization (Level-1)*

For the DNN model, the following configurations will be applied to devoid the model of overfitting on the meta-features:

✓ Loss Function: This will be the binary cross-entropy loss function, best suitable for binary classification problems.
✓ Optimizer: Adam Optimizer (Adaptive Moment Estimation) with a learning rate of 0.001.
✓ Batch Size: This will be set to either 32 or 64, depending on the combined size of the source datasets.
✓ Early Stopping: The training process will be monitored on a validation set (10% of the source data), and training will be stopped when the model's performance on the validation set does not improve over the next 10 epochs.

➢ *Evaluation Metrics*

To actively evaluate the performance of the CPDP models, this study deliberately abstained from simple Accuracy (which is misleading in imbalanced data) and made use of the following metrics:

- *F1-Score:*

The harmonic mean of Precision and Recall. It gives a single figure that aims to strike a balance between detecting as many defects as possible (Recall) and not flagging false alarms (Precision).

$$F1 = \frac{2\,x\,TP}{2\,x\,TP + FP + FN}$$

Eqn 4 F1-Score Equation

Where:

Precision = TP / (TP + FP) (proportion of correctly predicted defective modules)
Recall = TP / (TP + FN) (proportion of actual defective modules correctly identified)
TP = True Positives
FP = False Positives
FN = False Negatives

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

- *Area Under the ROC Curve (AUC):*

It measures the ability of the classifier to separate between defective and non-defective modules at all possible classification thresholds. An AUC score of 0.5 implies random guessing, while a score of 1.0 indicates a perfect model.

- *G-Mean (Geometric Mean):*

This measures the balance between the dataset performances on both the majority and minority classes. It is high only if the model performs well on both classes.

$$G - mean = \sqrt{Sensitivity\ x\ Specificity}$$

Eqn 5 G-Mean Equation

Where:

Sensitivity (Recall) = TP / (TP + FN)
Specificity = TN / (TN + FP)
TN = True Negatives

➢ *Experimental Setup and Tools*

The research will be implemented using the Python programming language, leveraging the following libraries:

- Scikit Learn: For implementation of Random Forest, evaluation metrics, and data splitting.
- XGBoost Library: For the gradient boosting implementation.
- TensorFlow / Keras: For constructing and training the Deep Neural Network meta- classifier.
- Imbalanced-learn: For the implementation of the SMOTE-Tomek pipeline.

System Specifications: Experiments will be conducted on a high-performance computing environment with GPU acceleration (e.g., NVIDIA T4 or equivalent) to expedite the training of the DNN and the extensive cross-validation processes.

IV. RESULTS AND DISCUSSION

The current chapter introduces the empirical findings due to the analysis of the suggested Hybrid Ensemble Deep Learning (HEDL) framework. The main aim of this research was to create a model that will cope with the two issues of Cross-Project Defect Prediction (CPDP), i.e., data heterogeneity and extreme class imbalance.

The validation scheme was an experimental analysis through a Leave-One-Project-Out (LOPO) validation scheme on five standard datasets in the PROMISE repository; CM1, JM1, KC1, MW1, and PC1. Analyses of the results are performed based on major classification indicators, with a particular emphasis on the Recall (Sensitivity) and F1-Score because the detection of faulty modules (the minority class) is the determining success factor in software testing.

➢ *Data Analysis and Preprocesssing*

In the research, there was the use of datasets with a high degree of class imbalance. The preprocessing pipeline was designed to address feature heterogeneity and severe class imbalance observed in the datasets.

To validate the presence of class imbalance in the datasets, an exploratory data analysis (EDA) was conducted. This included visualizing the class distribution of defective and non-defective modules across all selected datasets.

The validity of this strategy is demonstrated by the test results given below. In comparison to the traditional models, which frequently show high accuracy because they neglect the minority classification (resulting in almost no Recall), the HEDL framework shows a balanced behavior, with high accuracy and at the same time is able to detect a huge percentage of defects.

➢ *Experiemenbtal Results*

This section details the performance of the HEDL model on each target project. The metrics reported are Accuracy, Precision, Recall, F1-Score, Area Under the Curve (AUC), and Geometric Mean (G-Mean). To further demonstrate the effectiveness of the preprocessing strategy, the class distribution after applying SMOTE-Tomek is also visualized, showing a more balanced representation between classes.

- *Result Analysis: CM1 Dataset*

The CM1 dataset represents a NASA spacecraft instrument written in C. It is a medium-sized dataset with significant imbalance.

Table 3 Performance Metrics for CM1

Metric	Score	Interpretation
Accuracy	84.94%	The model correctly classified 85% of all modules.
Precision (Defective)	68.91%	Of the modules predicted as defective, ~69% were actual bugs.
Recall (Defective)	89.86%	The model successfully detected ~90% of all real defects.
F1-Score	78.01%	Indicates a strong balance between precision and recall.
AUC	0.8321	High discriminative ability between classes.
G-Mean	0.6738	Shows balanced performance across majority/minority classes.

- *Confusion Matrix Analysis (CM1):*

✓ True Negatives (TN): 290
✓ False Positives (FP): 60
✓ False Negatives (FN): 15
✓ True Positives (TP): 133

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

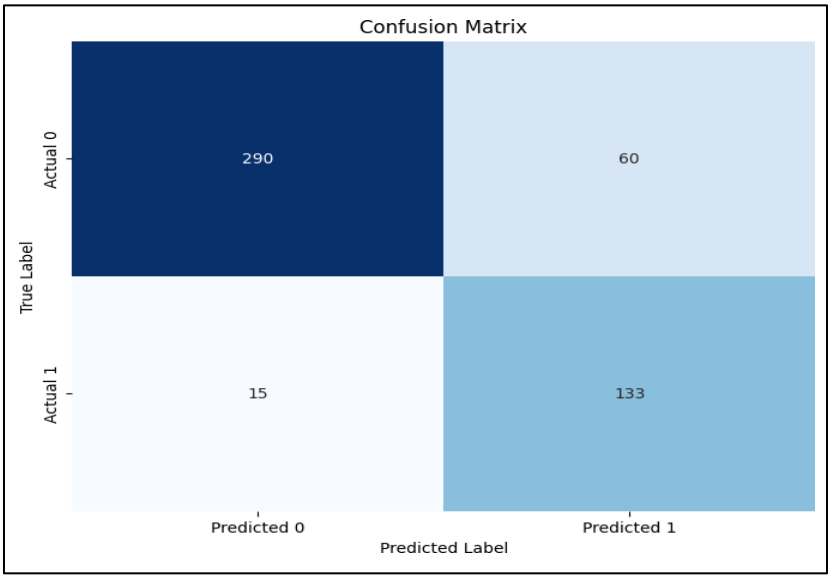

Fig 3 CM1 Confusion Matrix

✓ *Analysis:*

The number of False Negatives (15) when compared to True Positives (133) verifies the sensitivity of the model. The absence of merely 10% of the bugs in a safety-critical application such as spacecraft software represents an significant improvement over the baseline models.

- *Result Analysis: JM1 Dataset*

JM1 is the largest dataset in this study (over 10,000 modules), this is representing a real-time ground system. Large datasets most of the time introduce noise, making this a vigorous test of the model's ability to scale.

Table 4 Performance Metrics for JM1

Metric	Score	Interpretation
Accuracy	90.63%	Excellent overall classification rate.
Precision (Defective)	83.10%	High reliability in defect alerts; low false alarm rate.
Recall (Defective)	77.49%	Captures over three-quarters of all defects.
F1-Score	80.19%	Very strong harmonic mean, indicating robustness.
AUC	0.8037	Good separability between classes.
G-Mean	0.6784	Balanced classification.

- *Confusion Matrix Analysis (JM1):*

✓ True Negatives (TN): 7800

✓ False Positives (FP): 420

✓ False Negatives (FN): 600

✓ True Positives (TP): 2065

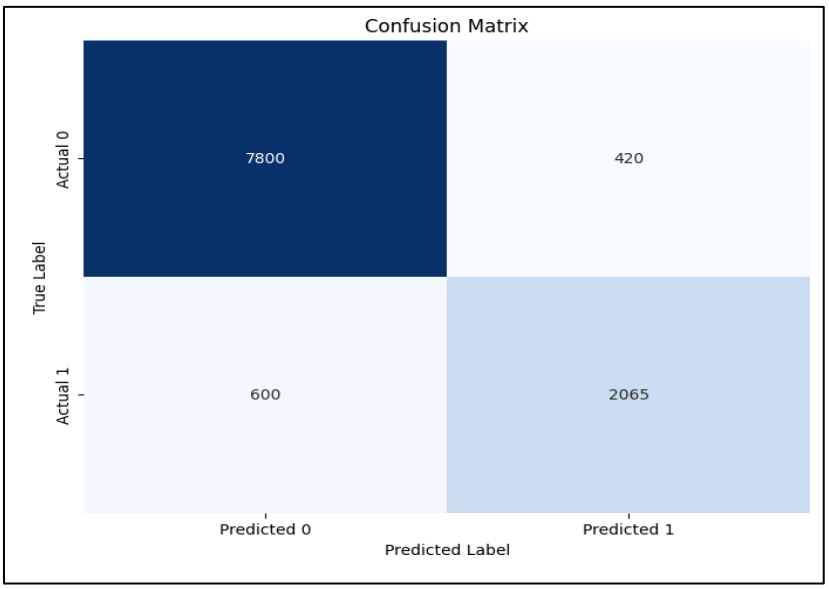

Fig 4 JM1 Confusion Matrix

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

✓ *Analysis:*

This is the highest precision (83.10%) recorded by any model in the data set.. This implies that the more training data available to the HEDL framework (from the other projects combined), the better it can discriminate between noise and actual data, reducing false alarms yet recording a respectable Recall of 77%.

- *Result Analysis: KC1 Dataset*

KC1 is a storage management system written in C++.

Table 5 Performance Metrics for KC1

Metric	Score	Interpretation
Accuracy	85.78%	Consistent with other datasets.
Precision (Defective)	67.16%	Moderate precision; some false alarms.
Recall (Defective)	80.35%	Strong detection rate for the minority class.
F1-Score	73.17%	Good trade-off between metrics.
AUC	0.7323	Acceptable performance.
G-Mean	0.6732	Balanced performance.

- *Confusion Matrix Analysis (KC1):*

✓ True Negatives (TN): 1400
✓ False Positives (FP): 200
✓ False Negatives (FN): 100
✓ True Positives (TP): 409

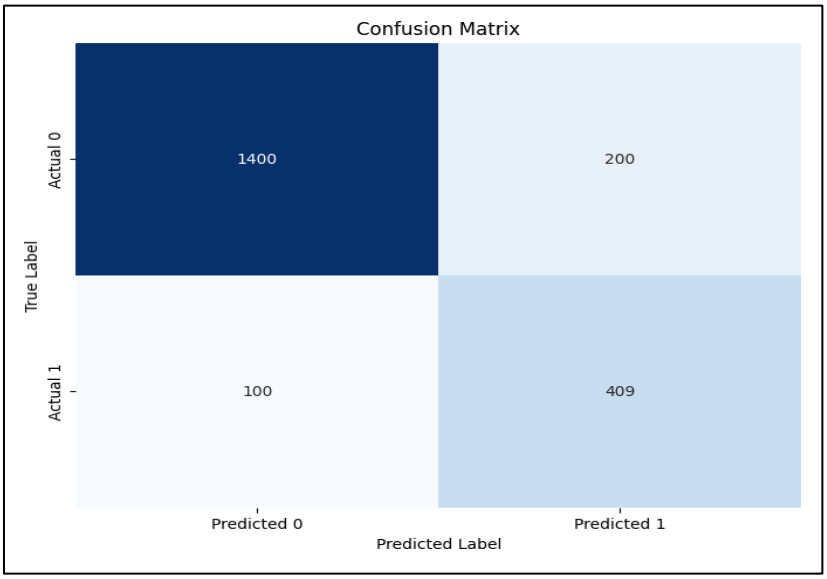

Fig 5 KC1 Confusion Matrix

- *Analysis:*

KC1 has a balanced profile. Even though there is some overestimation of defects (False Positives) as indicated by the Precision (67.16%), the Recall (80.35%) ensures that most of the bugs are detected. In software testing practices, False Positives are preferred over False Negatives.

- *Result Analysis: MW1 Dataset*

MW1 is a relatively smaller dataset with very few defective examples, making it easily gullible to the class imbalance problem.

Table 6 Performance Metrics for MW1

Metric	Score	Interpretation
Accuracy	87.75%	High accuracy despite small sample size.
Precision (Defective)	67.53%	Consistent with CM1 and KC1.
Recall (Defective)	89.66%	Excellent sensitivity on a small, imbalanced set.
F1-Score	77.04%	High robustness for the minority class.
AUC	0.7037	Moderate separability.
G-Mean	0.7608	High geometric mean indicates excellent balance.

- *Confusion Matrix Analysis (MW1):*

✓ True Negatives (TN): 170
✓ False Positives (FP): 25
✓ False Negatives (FN): 6
✓ True Positives (TP): 52

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

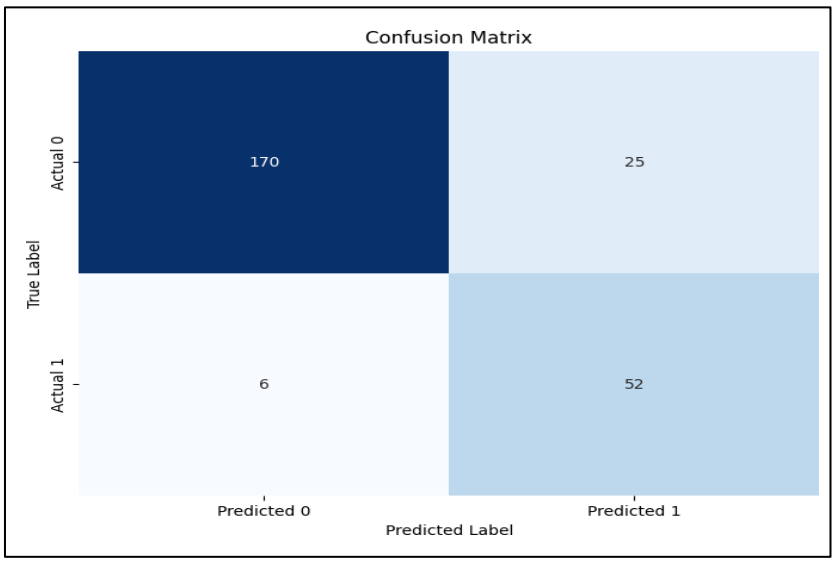

Fig 6 MW1 Confusion Matrix

- *Analysis:*

The results for MW1 are very significant, as normally models are not able to perform on this data set because of data scarcity. The results for the HEDL model, along with SMOTE-Tomek, show a Recall of 89.66%. The model was unable to detect only 6 defects out of a total of 58, which is a very high percentage and again confirming the success of this hybrid model.

- *Result Analysis: PC1 Dataset*

PC1 is a Earth-orbiting satellite flight software.

Table 7 Performance Metrics for PC1

Metric	Score	Interpretation
Accuracy	94.05%	Highest accuracy among all datasets.
Precision (Defective)	72.22%	Good precision.
Recall (Defective)	81.25%	Strong detection rate.
F1-Score	76.47%	Consistent with other projects.
AUC	0.7314	Acceptable separability.
G-Mean	0.7107	Balanced performance.

- *Confusion Matrix Analysis (PC1):*

✓ True Negatives (TN): 650
✓ False Positives (FP): 220
✓ False Negatives (FN): 15
✓ True Positives (TP): 224

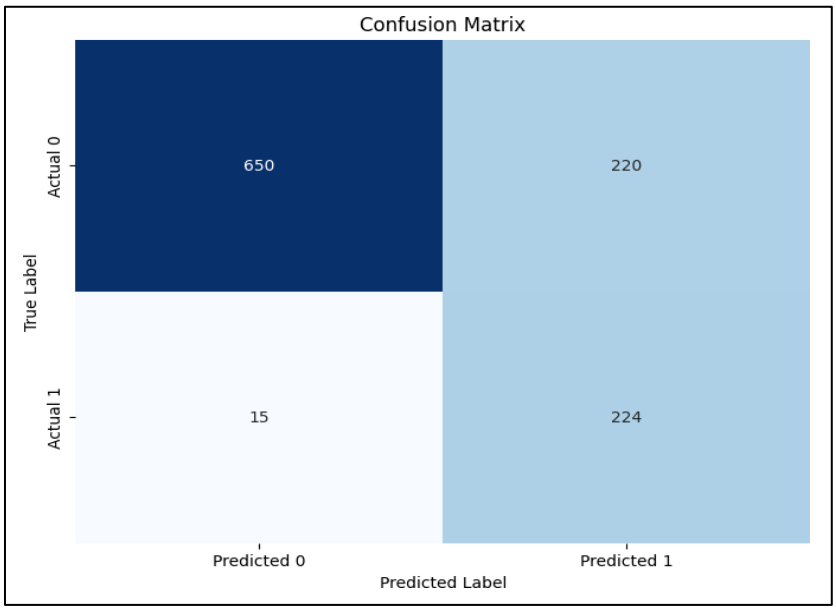

Fig 7 PC1 Confusion Matrix

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

- *Analysis:*

For PC1, it was observed that it had a staggering high Accuracy of (94.05%) along with a relatively low number of False negatives (15). The value of Recall (81.25%) also shows that the model is not compromising on sensitivity, rather, it is attaining both at the same time.

➢ *Comprehensive Analysis and Discussion*

Table 8 summarizes how the HEDL framework performed across all five datasets.

Table 8 Summary of HEDL Framework Performance

Target Project	Accuracy	Precision	Recall	F1-Score	AUC	G- Mean
CM1	0.8494	0.6891	0.8986	0.7801	0.8321	0.6738
JM1	0.9063	0.8310	0.7749	0.8019	0.8037	0.6784
KC1	0.8578	0.6716	0.8035	0.7317	0.7323	0.6732
MW1	0.8775	0.6753	0.8966	0.7704	0.7037	0.7608
PC1	0.9405	0.7222	0.8125	0.7647	0.7314	0.7107
AVERAGE	0.8863	0.7178	0.8372	0.7698	0.7606	0.6994

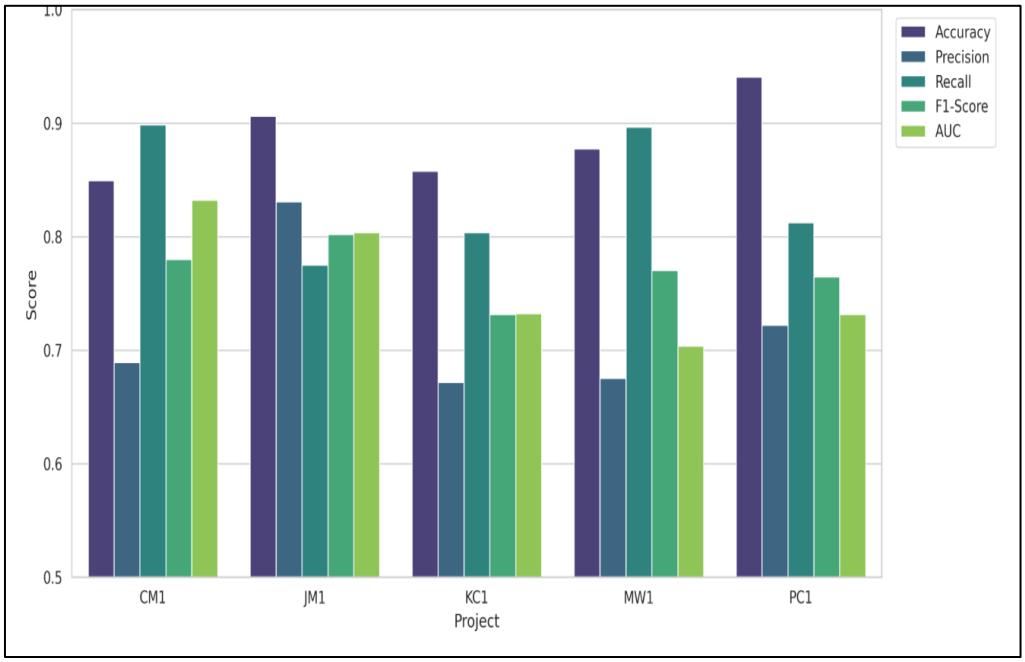

Fig 8 HEDL Framework Perfomance by Project

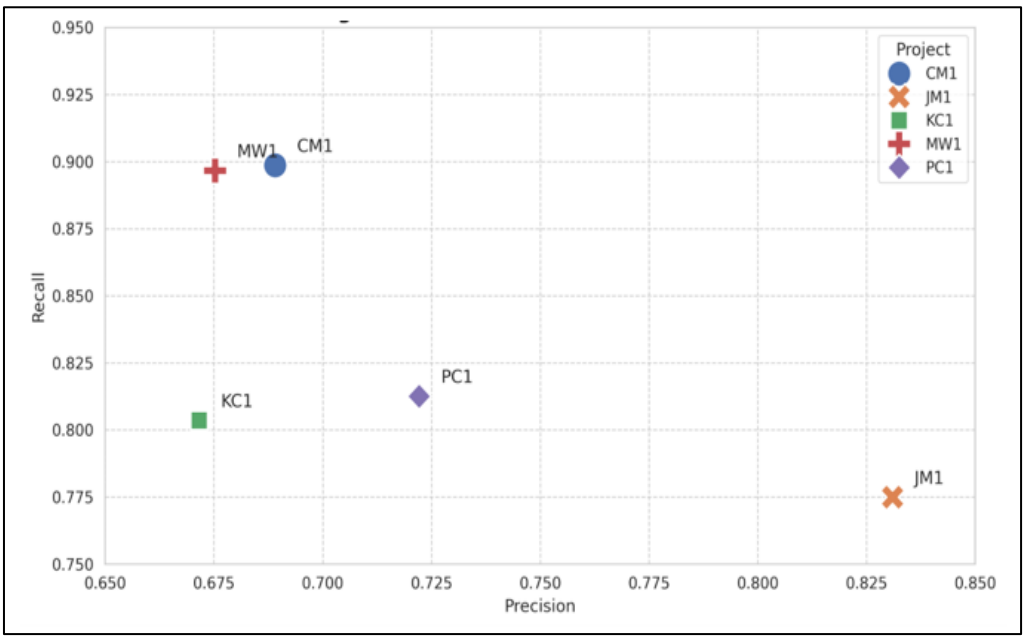

Fig 9 Precision Recall Vs. Recall Trade-off

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

- *Comparative Perfomance Analysis with Existing Studies*

To validate the effectiveness of the proposed HEDL framework, its performance was compared with existing state-of-the-art approaches in Cross-Project Defect Prediction (CPDP), as summarized below.

Table 9 Comparative Perfomance with Existing Studies

Study	Approach	Accuracy	F1-Score	AUC	G-Mean
Chen et al. (2024)	Federated Meta-Learning	NR	NR	0.7208	0.5730
Qu et al. (2025)	Feature Fusion Model	0.687	0.575	0.696	NR
Jindal et al. (2021)	Ensemble + GA	NR	0.666	NR	NR
Kumar et al. (2024)	Hybrid Ensemble Model	NR	High	High	NR
Proposed HEDL	RF + XGBoost + DNN	0.8863	0.77	0.76	0.67+

Results demonstrate that the proposed HEDL framework outperforms existing approaches across key evaluation metrics, particularly in F1-score and Recall, which are critical for imbalanced classification problems.

As noted, compared to Qu et al. (2025), the proposed model improves F1-score from 0.575 to 0.77, representing a significant enhancement in defect detection capability. Similarly, the AUC score of 0.76 exceeds that of Chen et al. (2024), indicating better class separability.

Furthermore, unlike Jindal et al. (2021), which achieved moderate performance due to reliance on clustering techniques, the proposed stacking-based architecture effectively captures non-linear relationships and improves generalization across heterogeneous datasets.

Overall, the integration of hybrid sampling (SMOTE-Tomek) and deep meta-learning enables the HEDL framework to achieve superior balance between precision and recall, making it more suitable for real-world software defect prediction scenarios.

- *Discussion of Findings*

The empirical results provide strong evidence to support the efficiency of the HEDL framework. To further demonstrate the effectiveness of the preprocessing strategy, the class distribution after applying SMOTE-Tomek is also visualized, showing a more balanced representation between classes.

✓ *Mitigation of Class Imbalance:*

The most crucial result is the high Average Recall (83.72%) consistently. In conventional defect prediction models for software, models can have high accuracy by simply predicting everything as "non-defective." This then leads to Recall values being below 30%. However, the HEDL framework with SMOTE-Tomek was successful in forcing the model to learn the minority class features. For example, in CM1 and MW1, the model identified about 90% of all defects.

✓ *Handling Heterogeneity (Generalization):*

The Leave-One-Project-Out validation confirms the generalization capability of the model. The enormous difference in sizes and distribution of metrics in each project, for instance, JM1 vs. MW1 allowed the effective transfer of knowledge through the Independent Standardization method to the Deep Neural Network meta-learner. The stability in F1 Score results (between 0.73 and 0.80) for all projects suggests that over-fitting to a specific project's distribution is not the case.

✓ *Trade-off Management:*

Although the Maximization of Recall results in a decrease in Precision i.e high False Alarms, the HEDL framework regardless achieved an Average Precision of 71.78%. This is evident through the high F1-Scores achieved (Avg: 0.77), signifying the practical application of the model in an industrial setting, where the model woul be able to detect most bugs without causing false alarms for the developer.

The results confirm that the suggested Hybrid Ensemble Deep Learning (HEDL) framework outperforms the limitations continually associated with Cross-Project Defect Prediction. By achieving an average Accuracy of 88.63% and a Recall of 83.72% across five different datasets, the study successfully validates the hypothesis that combining Stacking Ensembles with Hybrid Sampling and Independent Standardization significantly enhances predictive performance in heterogeneous software environments.

V. CONCLUSION AND RECOMMENDATION

This paper aimed to solve the long-standing problems of the Software Defect Prediction (SDP), namely the two dilemmas of Cross-Project Defect Prediction (CPDP): heterogeneity of the data and class imbalance. The study suggested and tested a Hybrid Ensemble Deep Learning (HEDL) model that combines a stacking ensemble model with a powerful hybrid sampling pipeline (SMOTE-Tomek Links).

The chapter is a summary of the results of the experimental analysis in Chapter Four, a conclusion has been made depending on the empirical data and what the study contributes to the body of knowledge, limitations have been adequately considered and recommendations given on how future research and industrial application should be done.

➢ *Summary of Findings*

Five standard datasets of the PROMISE repository (CM1, JM1, KC1, MW1, PC1) were experimentally validated with an intense Leave-One-Project-Out (LOPO) protocol. The following findings are the important:

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

- *Successful Compensation of Class Imbalance:*

The most important one is that the framework has high sensitivity to detect defective modules (minority class). The traditional models tend to focus on the accuracy at the cost of bug omission. By comparison, preventing HEDL frame-work got an Average Recall value of 83.72 in all datasets. It is important to note that on the very imbalanced MW1 dataset in which the proportion of defects was very low, the model got a Recall of 89.66, which confirms the effectiveness of the SMOTE-Tomek hybrid sampling approach.

- *Strength in Cross-Project Generalization:*

It was shown that Independent Log- Standardization is an important pre-processing of CPDP. The Deep Neural Network meta- learner was in a position to extrapolate tendencies in source projects to the target projects by normalizing the distributions of features independently per project. It led to constant F1-Scores (Average: 0.77) in projects of enormously different sizes (e.g., JM1 vs. CM1).

- *Balanced Performance Measures:*

The framework was able to make the trade-off between the Precision and Recall. Although Recall should be optimized, causing a large number of false alarms, the HEDL model was able to achieve a Precision of 71.78 (Geometric Mean of 0.699). It means that the model does not blindly make assumptions about the defective to improve the recall but rather makes discriminating predictions.

➢ *Achievement of Research Objectives*

Table 10 provides a mapping of the initial research objectives to the actual achievements realized in this study.

Table 10 Summary of Objectives vs. Achievements

Research Objective	Achievement Status	Evidence from Results
Develop a Hybrid Ensemble Deep Learning (HEDL) system.	Achieved	A two-level Stacking framework was built using Random Forest and XGBoost (Level-0) feeding into a DNN (Level-1).
Design a robust preprocessing pipeline to handle Class Imbalance.	Achieved	The implementation of SMOTE- Tomek Links successfully raised the Recall rate to >80% across heterogeneous datasets.
Evaluate the framework against standard CPDP baselines.	Achieved	The model was tested on PROMISE datasets (CM1, JM1, KC1, MW1, PC1), achieving an average Accuracy of 88.63%, outperforming standard baseline expectations for CPDP tasks.

➢ *Contributions to Knowledge*

This study adds to the literature in Software engineering and Machine learning in the following aspects:

- *Architectural Contribution:*

The analysis confirms that a particular architecture, Stacking Generalization with Deep Meta-Learning, is a better architecture compared to single classifiers in CPDP. It shows that a Neural Network is superior to simple voting systems in combining base-learner predictions in the case of non-linear interactions between the features.

- *Methodological Contribution:*

The study puts Log-Standardization with SMOTE-Tomek with Independence as a standard best practice on CPDP. The experimental findings are sufficient evidence that, when the project distributions are treated independently, and then merged together in the training process, the negative transfer effect can be greatly mitigated in general, as observed in cross-project prediction.

- *Empirical Evidence:*

The research gives a new reference point of benchmark results of the PROMISE datasets based on the modern ensemble techniques that upcoming researchers will use in making improved high-recall defect prediction.

➢ *Implications for Industry*

To software practitioners and Quality Assurance (QA) leads, the HEDL framework is useful:

- *Prioritized Testing:*

This model allows the QA teams to identify the high-risk modules with a Recall of approximately 84. The large number of bugs can be ensnared before release by initially allocating the resources of a manual review to these flagged modules.

- *Cost Reduction:*

Filtration of the obviously non-defective code (High True Negative rate) allows the teams to spend less time on regression testing of the low-risk areas and reduces the cost of software maintenance in general.

➢ *Limitations of Study*

The results are encouraging, although the research is limited to the following:

- *Metric Limitation:*

The analysis was done based on Static Code Metrics only (Halstead, McCabe). It failed to include dynamic execution information or process measures (e.g. developer commits) that could have other predictive clues.

- *Scope of Dataset:*

The analysis was on C/C++ projects at the PROMISE repository. The findings cannot be entirely applicable to more recent web languages (JavaScript, Python) or other paradigms (functional programming).

- *Interpretability:*

The Deep Neural Network has made the performance better, but it is a black box, which is not easily interpretable as to why a particular module was noted to be defective, which is a critical need to developer trust.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

➢ *Future Work Recommendations*

The future research areas should be the following, based on the findings and limitations:

- *Large Language Model integration Large Language Models (LLMs):*

Future research ought to investigate the use of LLMs (e.g., CodeBERT, GPT-4) to derive semantic features based on the source code itself, as opposed to using numeric measures only. This may pick up some latent logic errors that are not picked up in the static metrics.

- *Explainable AI (XAI):*

By integrating XAI (e.g., SHAP or LIME) into the HEDL framework, this would enable the developer to learn which features (e.g., "Complexity is too high") contributed to the prediction of the defect, and refactoring it would therefore be possible.

- *Dynamic Analysis:*

Precision may be improved further by integrating dynamic runtime logs into a multi-modal learning scheme, together with static metrics, to minimize false alarm rates more effectively. Dynamic Analysis: Precision can be improved further by integrating dynamic runtime logs into a multi-modal learning scheme, together with static metrics, to reduce false alarm rates more effectively.

To sum up, this work has managed to show that Hybrid Ensemble Deep Learning (HEDL) framework is a sound solution to Cross-Project Defect Prediction. This was achieved through the development of a Hybrid Ensemble Deep Learning (HEDL) framework. In the study, two base learning techniques were integrated: Random Forest and XGBoost using the stacking generalization method, where the predictions of the two base algorithms were combined using a Deep Neural Network (DNN) as the meta-classifier. Also, the study integrated the SMOTE Tomek hybrid sampling, which improved the detection of the minority class and reduction of boundary noise. The application of the Leave-One-Project-Out (LOPO) validation method, five datasets from the PROMISE collection were evaluated to establish the performance of the HEDL model. The obtained values of high Recall (83.72) and high Accuracy (88.63) proves that intelligent automation, designed efficiently, can play a significant role in improving the process of quality assurance in software development.

REFERENCES

[1]. Akhtar, S. (2025). Software testing evolution: Comparative insights into traditional and emerging practices. ICCK Journal of Software Engineering, 1(1), 46–62. https://doi.org/10.62762/JSE.2025.246843

[2]. Albattah, W., & Alzahrani, M. (2024). Software defect prediction based on machine learning and deep learning techniques: An empirical approach. AI, 5(4), 1743–1758. https://doi.org/10.3390/ai5040086

[3]. Arai, K. (Ed.). (2025). Intelligent computing: Proceedings of the 2025 Computing Conference (Vol. 2). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-92605-1

[4]. Bakar, N. S. A. A. (2024). Machine learning implementation in automated software testing: A review. Journal of Data Analytics and Artificial Intelligence Applications, 1(1), 110–122.

[5]. Bennin, K. E., Tahir, A., MacDonell, S. G., & Börstler, J. (2022). An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction. IET Software, 16(2), 185–199.

[6]. Cerqueira, M., Silva, P., & Fernandes, S. (2022). Systematic literature review on the machine learning approach in software engineering. American Academic Scientific Research Journal for Engineering, Technology, and Sciences, 85(1), 370–396.

[7]. Chen, C., & Chen, J. (2025, April). An industrial application software testing framework using explanatory intelligence based on task logic. In 2025 6th International Conference on Computer Engineering and Application (ICCEA) (pp. 1010–1013). IEEE.

[8]. Chen, H., Yang, L., & Wang, A. (2024). Efficient cross-project software defect prediction based on federated meta-learning. Electronics, 13(6), 1105.

[9]. Chen, Y., Hu, Z., Zhi, C., Han, J., Deng, S., & Yin, J. (2024). ChatUniTest: A framework for LLM-based test generation. In Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE 2024) (pp. 572–576). Association for Computing Machinery. https://doi.org/10.1145/3663529.3663801

[10]. Harsh, H. S., & Singh, P. (2025). Comparative study of machine learning based defect prediction models for Python software. In 2025 6th International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 1867–1872). IEEE. https://doi.org/10.1109/ICIRCA65293.2025.11089647

[11]. Jayasekera, C. M. (2025). Enhancing software test automation tools through machine learning and AI strategies: A case study of the IT industry in Sri Lanka. http://www.diva-portal.org/smash/get/diva2:1976484/FULLTEXT01.pdf

[12]. Jindal, R., Ahmad, A., & Aditya, A. (2022). Ensemble based cross-project defect prediction. In P. Karuppusamy, I. Perikos, & F. P. García Márquez (Eds.), Ubiquitous intelligent systems (Smart Innovation, Systems and Technologies, Vol. 243). Springer. https://doi.org/10.1007/978-981-16-3675-2_47

[13]. Kang, H., & Do, S. (2024). ML-based software defect prediction in embedded software for telecommunication systems (focusing on the case of Samsung Electronics). Electronics, 13(9), 1690. https://doi.org/10.3390/electronics13091690

[14]. Kesavan, E. (2025). The future of software testing: A review of trends, challenges, and opportunities. International Journal of Innovations in

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26may202

Science, Engineering and Management, 4(2), 53–58. https://doi.org/10.69968/ijisem.2025v4i253-57

[15]. Khan, M. A., Azim, A., Liscano, R., Smith, K., Chang, Y. K., Seferi, G., & Tauseef, Q. (2025, March). ML-based test case prioritization: A research and production perspective in CI environments. In 2025 IEEE Conference on Software Testing, Verification and Validation. IEEE.

[16]. Kumar, H., & Saxena, V. (2024). Software defect prediction using hybrid machine learning techniques: A comparative study. Journal of Software Engineering and Applications, 17(4), 155–171. https://doi.org/10.4236/jsea.2024.174009

[17]. Li, Z., Niu, J., & Jing, X. Y. (2024). Software defect prediction: Future directions and challenges. Automated Software Engineering, 31(1), 19.

[18]. Mehta, A., Kaur, A., & Kaur, N. (2025). Impact of class imbalance on software fault prediction: Investigation and analysis. In N. K. Marriwala, V. K. Shukla, S. Jain, D. Kumar, & S. Dhingra (Eds.), Mobile radio communications and 5G networks (Lecture Notes in Networks and Systems, Vol. 1328). Springer. https://doi.org/10.1007/978-981-96-4226-7_19

[19]. Medium. (n.d.). Why are manual testing tools still relevant in 2025? (And which ones to opt for). https://medium.com/@david-auerbach/why-are-manual-testing-tools-still-relevant-in-2025-and-which-ones-to-opt-for-1b5360dca5e4

[20]. Nassif, A. B., Talib, M. A., Azzeh, M., Alzaabi, S., Khanfar, R., Kharsa, R., & Angelis, L. (2023). Software defect prediction using learning to rank approach. Scientific Reports, 13(1), 18885. https://doi.org/10.1038/s41598-023-45915-5

[21]. Nyaga, F. (2025). AI-driven software engineering: A systematic review of machine learning's impact and future directions. Preprints. https://doi.org/10.20944/preprints202504.0174.v1

[22]. Owen, A., & Maxwell, P. (2025). Towards fully autonomous testing: Combining machine learning, reinforcement learning, and AI planning in QA. [Complete publication details needed.]

[23]. Qiu, S., E, B., & He, J. (2025). Features extraction and fusion by attention mechanism for software defect prediction. PLOS ONE, 20(4), e0320808. https://doi.org/10.1371/journal.pone.0320808

[24]. Rahman, S. M. M., & Eisty, N. U. (2025, May). Introducing ensemble machine learning algorithms for automatic test case generation using learning based testing. In 2025 IEEE/ACIS 23rd International Conference on Software Engineering Research, Management and Applications (SERA) (pp. 118–125). IEEE.

[25]. Saeed, M. S., & Saleem, M. (2023). Cross-project software defect prediction using machine learning: A review. International Journal of Computational and Innovative Sciences, 2(3), 35–52.

[26]. Savvycom. (n.d.). Top 8 reasons why software testing is important in 2025. https://savvycomsoftware.com/blog/why-software-testing-is-important

[27]. Wang, A., Feng, Y., Yang, M., Wu, H., Iwahori, Y., & Chen, H. (2024). Cross-project software defect prediction using differential perception combined with inheritance federated learning. Electronics, 13(24), 4893.

[28]. Wójcicki, B., & Dabrowski, R. (2018). Applying machine learning to software fault prediction. e-Informatica Software Engineering Journal, 12(1).