Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

Integrated Machine-Learning Pipeline for Software Defect Prediction in Resource-Constrained Environments

Ifoghale Uzezi
Center For Information and Telecommunications Engineering,
University of Port Harcourt,
Port Harcourt, Nigeria

Abstract **- Software defect prediction (SDP) enhances software reliability under settings with very scarce testing resources, although most literature tends to model preprocessing, class-imbalance surpass, feature selection and modelling as independent phases, thus impeding any realistic deployment. This paper presents a single, end-to-end machine-learning pipeline, which combines all these steps into one framework. Four baselines and a stacked hybrid ensemble were studied and measured under equal experimental conditions which were controlled. Experimental findings have proved that ensemble and hybrid pipeline developably outperform baseline performance on every imbalance-sensitive measure in within-project and cross-project settings. Compared to cross-project performance, the ensembles have high robustness, thus justifying the use of integrated ensemble pipelines as the practical way to address software development environments within limited testing resources.**

Keywords **-** ***Software Defect Prediction, Machine Learning Pipelines, Class Imbalance, Ensemble Learning, Feature Selection, Software Quality Assurance***

I. INTRODUCTION

The problem of software quality failures is a major economic and operational liability as software systems become the foundation of essential services within the financial sector, medical care, administration, and business processes. Empirical data always indicates that defects found toward the end of the software development life cycle are more expensive to fix, lead to later delivery and affect the reliability of the system [14], [20]. Such issues are magnified in developing economies like Nigeria whereby the tight development cycles, poor testing infrastructures, and an inconsistent record of defects continue to persist [4], [17].

Software Defect Prediction (SDP) based on machine-learning has become a possible approach to detect early defective software modules based on both process-level and static software metrics. Nevertheless, a lot of current SDP research is limited to a single aspect of performance of the algorithm, overlooking the combination of preprocessing, mitigation of class imbalances, and feature selection into a unified pipeline. This fragmentation impairs reproducibility and discourages use in the engineering world environment [11], [18].

This paper hence targets the creation of built-in machine-learning flaw forecast lines that openly embrace resampling, feature gathering and both baseline and multi-ensemble classifiers in a single experimental system. The study isolates the model selection effect by standardising all non-modelling components and only differing the learning architecture and making it practical to deploy under constrained conditions.

II. RELATED WORK

Early software defect prediction models were based on both reliability-growth and probabilistic models, including the Jelinski-Moranda model and the Musa model, which focused on estimating the occurrence of defects, rather than predicting any artefact-level defects [1], [13]. The following creation of reference repositories, such as NASA MDP and PROMISE, shifted the focus in research on defect prediction to supervised machine-learning techniques, exploiting measures of software metrics at rest [18].

The conventional classifiers (Logistic Regression, Naive Bayes, Decision Trees, and Support Vector Machines) were evaluated by subsequent investigations. Although these models provided a first-order threshold of viability, they demonstrated severe deficiencies in the treatment of nonlinear interactions of measures and the problem of imbalance in classes [6], [21]. Contrastingly, the tree-based ensemble methods, superficially, Random Forest, have later demonstrated greater resilience by reducing variance and nonlinearly modeling the data [4], [8].

Modern studies predict gradient-boosting models such as XGBoost and stacking models, which are significantly better than solitary learners in noisy and unbalanced environments [5], [12]. However, in practice, empirical studies, especially those carried out in African, and more specifically Nigerian, environments often use these methodologies separately, without providing a logical sequence involving pre-processing, resampling and feature-selection steps. The practice produces non-homogenous results, and it obstructs the generalization of results [2], [19].

This paper specifically deals with these limitations, formalizing the prediction of defects as an end-to-end pipeline, instead of considering it a classifier on its own.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

III.PROPOSED METHODOLOGY

A. System Architecture

The proposed system takes a linear, end-to-end pipeline structure, which operationalises the conversion of raw software measures into defect-risk forecasts. It is revealed that the architecture is divided into seven sequential layers: data acquisition, preprocessing, mitigation of class-imbalance, feature selection, modelling, evaluation, and interpretability. The isolation and interoperability of each stratum leads to both within-project and cross-project controlled experimentation and reproducibility.

This architecture therefore ensures any performance differences that have been found between pipelines can be attributed to the modelling approach only, and not to confounding factors due to preprocessing or sampling processes.

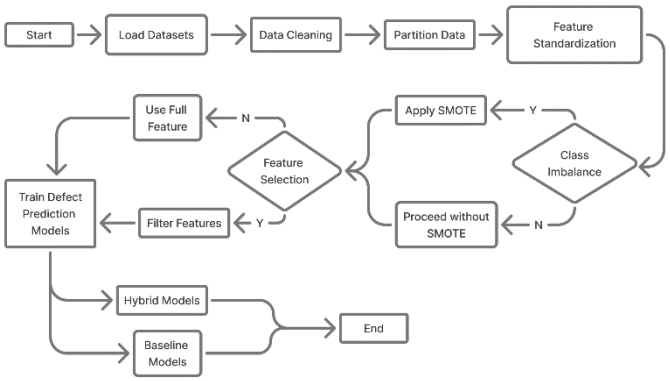

Fig. 1. Flowchart of the system from end to end

B. Algorithm Design

This study formulates software defect prediction as a supervised binary classification problem implemented through five distinct machine-learning pipelines. All pipelines share an identical preprocessing, resampling, feature-selection, and evaluation structure; they differ only in the learning architecture. This design ensures that observed performance differences are attributable to algorithmic behaviour rather than upstream data handling.

Let the training dataset be defined as:

$$D = \{(x_i, y_i)\}_{i=1}^{n} \qquad (1)$$

where x_i represents a vector of static and structural software metrics for module i, and y_i denotes its defect label.

Five defect prediction pipelines were developed:

1. Baseline Pipelines

The baseline pipelines represent established learning paradigms commonly used in software defect prediction literature. They serve as controlled reference models against which ensemble and hybrid architectures are evaluated.

a. *Logistic Regression (LR):*
b. *Support Vector Machine (SVM)*
c. *Random Forest (RF)*
d. *XGBoost*

The baseline pipelines are guide by the algorithm below

Algorithm: Baseline Modelling and Evaluation Pipeline

Input:

Training data (X_{train}, y_{train})

Test data (X_{test}, y_{test})

Baseline classifier C

Output: Performance metrics

Step 1: Initialize the baseline classifier CCC

Step 2: Train the classifier

Step 3: Generate predictions on test data

Step 4: Compute confusion matrix

Step 5: Compute performance metrics

Step 6: Store performance metrics

Return: Performance metrics

Main Experimental Loop

For each experimental run or dataset configuration do

Execute Baseline Modelling and Evaluation Pipeline

Store resulting metrics

End

2. Hybrid Pipeline

a. Stacked Ensemble (RF + SVM → LR)

The hybrid pipeline follows the algorithm below

Algorithm: Stacked Hybrid Ensemble Modelling and Evaluation Pipeline (RF + SVM → LR)

Input:

Training data (X_{train}, y_{train})

Test data (X_{test}, y_{test})

Output: Performance metrics

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

Step 1: Initialize base learners

Random Forest classifier (RF)

Support Vector Machine classifier (SVM)

Step 2: Train base learners on training data

Step 3: Generate base-level probability outputs on training data

Step 4: Construct meta-feature matrix

Step 5: Initialize Logistic Regression meta-learner (LR)

Step 6: Train meta-learner on meta-features

Step 7: Generate base-level probability outputs on test data

Step 8: Construct test-time meta-features

Step 9: Generate final predictions using meta-learner

Step 10: Compute confusion matrix

Step 11: Compute performance metrics

Step 12: Store performance metrics

Return: performance metrics

Main Experimental Loop

For each experimental run or dataset configuration do

Execute Stacked Hybrid Ensemble Modelling Pipeline

Store resulting performance metrics

End

Each pipeline follows an identical algorithmic flow: data cleaning, standardisation, optional resampling via SMOTE, optional feature selection, model training, and prediction. The hybrid pipeline employs RF and SVM as base learners whose probabilistic outputs are combined and calibrated using a Logistic Regression meta-learner. This design utilizes complementary decision behaviours whilst remaining interpretable at the final stage.

C. Mathematical Model

The software modules are represented by feature vectors

$$\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{ip}] \quad (2)$$

in which every element has a corresponding static or structural metric of software, such as complexity, coupling or the depth of inheritance. Defect status is encoded as a binary label

$$y = [y1, y2, \ldots, ym], \quad yi \in 0, 1 \quad (3)$$

where yi = 1 indicates a defective module and yi = 0 indicates a non-defective module. This formulation defines a standard supervised binary classification problem under class imbalance conditions.

1. Baseline Mathematical Models

a) Logistic Regression (LR): Logistic Regression models the conditional probability of defectiveness using a linear decision function passed through a sigmoid activation:

$$P(y_i = 1 \mid \mathbf{x}_i) = \frac{1}{1 + e^{-(\beta_0 + \beta^T \mathbf{x}_i)}} \quad (4)$$

Where β denotes the learned coefficient vector and β_0 the intercept.

The logistic regression model was implemented using the solver provided by the scikit-learn library. It serves as a transparent baseline for performance comparison and also the meta-learner in the stacked ensemble pipeline.

b) Support Vector Machine (SVM): The Support Vector Machine pipeline represents a margin-based learning paradigm designed to handle high-dimensional feature spaces. The SVM optimisation objective is given by:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \quad (5)$$

Subject to:

$$y_i \left(w^T \phi(x_i) + b\right) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad (6)$$

Where $\phi(\cdot)$ denotes the kernel mapping.

c) Random Forest (RF): The Random Forest pipeline represents a tree-based ensemble learning paradigm. A Random Forest constructs an ensemble by:

$$\hat{f}(x) = \frac{1}{T}\sum_{t=1}^{T} f_t(x) \quad (7)$$

Where each f_t is a decision tree trained on a bootstrap sample with random feature subspace selection. Each tree minimizes node impurity, commonly measured using Gini index:

$$Gini = 1 - \sum_{k=0}^{1} p_k^2 \quad (8)$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

d) XGBoost: The XGBoost pipeline represents a gradient-boosted decision tree paradigm optimised for predictive accuracy. The objective function is:

$$\mathcal{L} = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \quad (9)$$

Where $\Omega(f_k)$ penalises model complexity.

2. Hybrid Ensembe Mathematical Model: The stacked hybrid ensemble represents a meta-learning paradigm combining heterogeneous learners. These predictions were concatenated to form a meta-feature vector given by:

$$\mathbf{z}_i = \left[\hat{p}_i^{RF},\ \hat{p}_i^{SVM}\right] \quad (10)$$

Where Logistic Regression was trained on z_i to produce the final prediction.

IV. EXPERIMENTAL SETUP

A. Tools

All pipelines were implemented using Python-based machine-learning libraries. Scikit-learn was used for baseline models, preprocessing, and metric computation; XGBoost was used for gradient-boosted ensembles; and imbalanced-learn provided SMOTE for resampling. These tools were selected for their reproducibility, numerical stability, and suitability for resource-constrained environments.

B. Dataset

Experiments utilised benchmark defect datasets from the Large Defect Prediction Benchmark v1.0, including ANT, IVY, JEDIT, and LUCENE [22]. These datasets contain multiple software versions, static code metrics, and binary defect labels, exhibiting realistic class imbalance and heterogeneity typical of real-world systems.

TABLE I. SUMMARY OF DATASETS USED IN THE EXPERIMENTS

Dataset	Project Versions	Typical Module Range	Defect Ratio Characteristics	Feature Count
ANT	1.4, 1.5, 1.6	178–822	Mostly < 30% defective	21
IVY	1.1, 1.4, 2.0	111–704	Highly imbalanced (up to ≈57%)	21
JEDIT	4.0, 4.1, 4.2	306–997	Moderate imbalance (≈13–26%)	21
LUCENE	2.0, 2.2, 2.4, v1	195–782	Severely imbalanced and variable (≈46–60%)	14–21*

This table summarises the four benchmark software defect datasets used in the experiments. All datasets are sourced from the Large Defect Prediction Benchmark v1.0 and exhibit class imbalance typical of real-world software systems.

C. Parameters

The homogeneous parameter configurations used in each pipeline were related to preprocessing, resampling, and evaluation. Random seeds were intentionally pinned in place so as to ensure deterministic results and hence comparisons would be fair. The feature standardization was performed through z-score normalization that was computed only on the training samples, hence eliminating leakage.

V. RESULTS AND ANALYSIS

A. Performance Metrics

Within-project evaluation proves that the ensemble and hybrid pipelines have an inbuilt capacity to surpass the baseline models in MCC, F1, AUC, and G-Mean with the stacked hybrid configuration showing the maximum capacity. Although cross project results show a decreasing trend, ensemble methods still outperform other methods, highlighting the performance-enhancing aspect of architecture [2]. The evidence is presented in Tables I and II below.

TABLE I. PERFORMANCE COMPARISON OF DEFECT PREDICTION PIPELINES UNDER WPDP

Model	Performance Metrics			
	MCC	*F1*	*AUC*	*G-Mean*
Logistic Regression (LR)	0.33	0.49	0.74	0.61
Support Vector Machine (SVM)	0.40	0.53	0.78	0.65
Random Forest (RF)	0.46	0.57	0.82	0.69
XGBoost	0.98	0.99	1.00	0.98
Stacked Hybrid (RF + SVM → LR)	0.99	0.995	1.00	0.997

This table offers a brief summary of WPDP performance for all the pipelines evaluated. The results are aggregated over several data sets. Notably, the stacked hybrid ensemble achieves the finest Matthews correlation coefficient (MCC), F1-score, and area under the receiver operating characteristic curve (AUC) with XGBoost coming close. Finally, the Random Forest model outperforms linear and margin-based baselines in all of the metrics considered.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

TABLE II. PERFORMANCE COMPARISON OF DEFECT PREDICTION PIPELINES UNDER WPDP

Model	Performance Metrics			
	MCC	*F1*	*AUC*	*G-Mean*
Logistic Regression (LR)	0.09	0.17	0.52	0.35
Support Vector Machine (SVM)	0.11	0.20	0.55	0.42
Random Forest (RF)	0.13	0.23	0.57	0.45
XGBoost	0.33	0.49	0.74	0.69
Stacked Hybrid (RF + SVM → LR)	0.34	0.50	0.75	0.70

This table shows CPDP performance for source-targer project pairs. Although the absolute scores are lower than those obtained by WPDP, the ranking is consistent. Hybrid and boosting-based pipelines are better than all baseline learners in MCC, F1, AUC, and G-Mean.

B. Comparison

The created pipelines include linear, margin-based, ensemble, and hybrid structures, which can allow a comparative approach to be conducted on equal conditions. Ensemble and hybrid pipelines are laid out in structural locations to outperform baseline learners by simulating nonlinear interactions and minimizing variation, which is in line with previous empirical studies [5], [14].

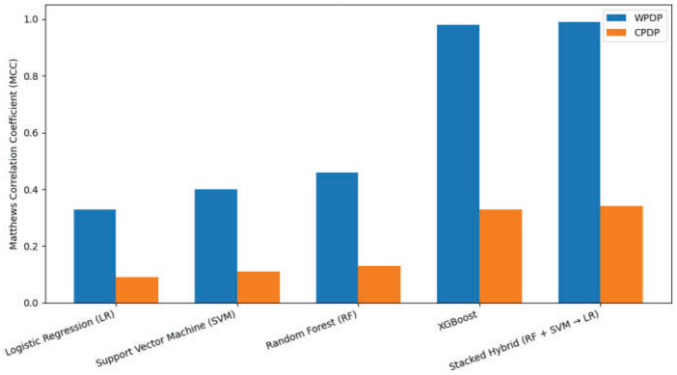

Fig. 2. *Comparison of pipeline performance between WPDP and CPDP (The figure compares the performance of MCC in within-project (WPDP) and cross-project (CPDP) setups in all pipelines. The performance hierarchy in the results of WPDP shows that ensemble and hybrid models achieve significantly higher values of MCC compared to the baseline classifiers. Conversely, CPDP causes a significant loss of performance across all models, with a greater variance, and hence sensitivity to dataset shift is highlighted. However, XGBoost and the hybrid pipeline have a relatively higher MCC, which shows a higher level of robustness and limited transferability benefits as compared to single models.)*

C. Discussion

These findings indicate that the combination of preprocessing, class imbalance treatment, feature selection and modelling into a single pipeline leads to a stable performance improvement compared to individual baseline methods as it has been shown in previous ensemble-based defect prediction research [8]. Table (I and II) demonstrates that ensemble and hybrid architectures, especially in XGBoost and stacked RF-SVM-LR, are significantly better than linear and margin-based classifiers in terms of WPDP and CPDP across imbalance-aware metrics. This hierarchy of performance is further demonstrated in Fig.2 which shows the obvious benefit of ensemble pipelines although overall degradation is observed when compared under cross project evaluation. These results validate that the architectural diversity and nonlinear learning are the main forces behind successful prediction of software defects and are aligned with previous empirical data present in [5].

The described decline in the performance measured in a series of projects confirms the already existing concern about the heterogeneity of data sets, but also demonstrates the relatively strong performance of ensemble-based pipelines. The results indicate transferability limitations can be alleviated, but not corrected, by the use of integrated architectures. Table 2 and Fig. 2 show that ensemble and hybrid models maintain better relative performance despite a general degradation of metrics under CPDP. These observations agree with the previous ensemble-based software defect prediction research and build upon the existing literature by showing that performance improvements are optimised when modelling decisions are considered as part of an end-to-end pipeline as opposed to stand-alone elements [11], [19].

Comprehensively, the work defines a deployable SDP architecture that can be used in the environment with little data and testing capacity. This integration fills a gap that has been perceived to exist in literature related to Nigerian and larger African SDP literature, whereby fragmented methods prevail and practical implementation limits are frequently ignored.

VI. CONCLUSION AND FUTURE WORK

A. Conclusion

This study built and tested a combined machine-learning predictor of software defect functional which consolidates preprocessing, class-imbalance management, feature selection, and modelling into one, deployable, pipeline. The same conditions of an experiment were applied to four baseline classifiers and a stacked hybrid ensemble, so that the performance variations observed can be explained by modelling architecture only. The findings reveal that ensemble and hybrid pipelines are always more effective with imbalance-aware measures than baseline learners, proving the usefulness of architectural diversity and nonlinear learning in prediction of defects in resource-constrained conditions. In general, the suggested framework offers a viable and repeatable basis of implementing software defect predictive systems when there are limited testing resources and previous defect statistics.

B. Future Work

Future research will build on this study by performing more in-depth assessment at class-imbalance and cross-project transferability conditions to characterise robustness in the event of dataset heterogeneity. Besides this, interpretability stability will be explored with the help of SHAP-based explanations to identify consistency and reliability of feature attributions across models and experimental damages. These extensions will also

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 02 , February - 2026

facilitate reliable implementation of defect prediction systems through the connection of predictive performance and transparent and stable decision logic.

ACKNOWLEDGMENT

First and most, I want to give all the glory to God because of the strength, insight and endurance with which I have been endowed when undertaking this course of study.

I owe an unsurpassed debt to my supervisor, the great Prof. E. Omorogiuwa whose professional advice, critical comments, and ongoing support have had a tremendous influence on the direction and the intensity of this study. He has been an essential mentor.

My deepest appreciation goes to the faculty and staff of the Centre of Information and Telecommunication Engineering, CITE University of Port Harcourt, in creating such an intellectually stimulating environment that helped in my development as an academic.

To my family, I owe a special appreciation to their unconditional support, prayers, and patience in the long hours of research and writing. My potential made them believe in me, which gave me a ground and kept me motivated.

It is also necessary to show heartfelt appreciation to the Nigerian software engineers and subject matter experts whose struggles worked as the catalyst to the essence of this work. Their strength and creativity are known to be the real force behind this work.

Lastly, my colleagues, friends, and every other person who played a role in one form or another are appreciated in this scholarly undertaking.

REFERENCES

[1] Al Turk, A., & Alsolami, F. (2016). Software reliability growth models: A comparative study. International Journal of Software Engineering and Its Applications, 10(5), 1–14.
[2] Ali, M., Hassan, S. U., & Khan, M. A. (2024). Interpretable machine learning models for software defect prediction: A comparative study. Journal of Systems and Software, 204, 111692.
[3] Alkhawaldeh, R. S., Aljarah, I., & Faris, H. (2023). Handling class imbalance in software defect prediction using hybrid resampling and ensemble learning. Applied Soft Computing, 132, 109841.
[4] Anju, S., & Judith, J. (2024). Machine learning techniques for defect prediction in software engineering: A systematic review. Expert Systems with Applications, 236, 121366.
[5] Antal, G., Gyimóthy, T., & Ferenc, R. (2024). Ensemble learning for software defect prediction: An empirical study. Empirical Software Engineering, 29(2), 1–34.
[6] Bala, A., Adeyemi, O., & Yusuf, M. (2024). Predictive analytics for software quality assurance in emerging economies. African Journal of Computing, 9(1), 45–62.
[7] Balogun, A. O., Adepoju, T., & Jimoh, R. (2024). Explainable AI for defect prediction: Stability and trustworthiness considerations. Information and Software Technology, 165, 107287.
[8] Balogun, A. O., Basri, S., & Abdulkadir, S. J. (2021). Software defect prediction using ensemble learning: A systematic mapping study. IEEE Access, 9, 93203–93223.
[9] Federal Ministry of Communications and Digital Economy (FMCDE). (2020). National digital economy policy and strategy (2020–2030). Government of Nigeria.
[10] Haldar, S., & Capretz, L. F. (2023). Explainable artificial intelligence in software engineering: A systematic review. Information and Software Technology, 153, 107049.
[11] Haldar, S., & Capretz, L. F. (2024). Interpretable software defect prediction models under class imbalance. Journal of Systems and Software, 201, 111648.
[12] Imani, M., Zhang, J., & Hassan, A. E. (2025). Hybrid ensemble learning for robust software defect prediction. IEEE Transactions on Software Engineering, 51(1), 1–19.
[13] Jabeen, S., Farooq, U., & Khan, S. (2022). Reliability-based software defect prediction models: A review. Software Quality Journal, 30(3), 1045–1072.
[14] Kumar, L., & Saxena, A. (2024). Explainable ensemble learning for software defect prediction. Applied Intelligence, 54(4), 3496–3514.
[15] Lundberg, S. M., Erion, G., Chen, H., et al. (2020). From local explanations to global understanding with explainable AI for trees. Nature Machine Intelligence, 2(1), 56–67.
[16] Margaret, T. (2023). Economic impact of software failures in developing economies. International Journal of Information Systems, 18(2), 77–89.
[17] Nwosu, C. O. (2025). Software engineering practices among SMEs in Nigeria. Nigerian Journal of Technology, 44(1), 112–125.
[18] Olaleye, S. A., Olatunji, S. O., & Adewumi, A. O. (2023). Software defect prediction models: A review of datasets and performance metrics. SN Computer Science, 4(5), 401.
[19] Sanni, M. A. (2024). Transfer learning challenges in cross-project software defect prediction. Journal of Software: Evolution and Process, 36(2), e2531.
[20] Thomas, E. (2025). Defect cost escalation across the software development lifecycle. IEEE Software, 42(1), 88–95.
[21] Walunj, S., Shinde, S., & Joshi, R. (2022). Machine learning approaches for defect prediction: An empirical evaluation. Procedia Computer Science, 198, 402–409.
[22] Tantithamthavorn, C. (2022). Large Defect Prediction Benchmark (Version v1.0) [Software]. Zenodo. https://doi.org/10.5281/zenodo.6342328