Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

Testing of software projects based on function points through a machine-learning approach

Hemant Kumar*
Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Vidya Vihar, Rae Bareli Road, Lucknow 226025, India
Email: hemant20192@gmail.com
Vipin Saxena
Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Vidya Vihar, Rae Bareli Road, Lucknow 226025, India
Email: profvipinsaxena@gmail.com

ABSTRACT

The software projects are coded by the software engineers either through designing function points or lines of code techniques. Both methods are equally important but due to the evolution of object-oriented technology. Function points are preferred concerning lines of code for medium and long software projects. Further, the machine learning approach is useful for testing software projects based on the Python programming language. The present paper is based on the machine learning-based testing strategy to be used for long software projects written in Python by the software industries. The computed reports are depicted in the form of tables and graphs.

Keywords – **Function points, Lines of code, Machine learning, Python programming, Software projects, Testing.**

Date of Submission: January 17, 2025 Date of Acceptance: February 25, 2025

I. INTRODUCTION

In the realm of software engineering, the development and testing of software projects are pivotal stages that determine the success and reliability of the final product. Traditional methodologies for measuring software size and complexity have evolved, with function points emerging as a prominent metric alongside lines of code. Function points offer a more abstract and comprehensive measure of software functionality, making it particularly suitable for medium and long-term projects, especially in the context of object-oriented programming. As software projects grow in complexity and scale, the need for efficient testing strategies becomes increasingly paramount. In recent years, machine learning has emerged as a powerful tool in various domains, including software engineering, offering the potential to enhance testing processes and improve project outcomes. By leveraging machine learning algorithms, software engineers can analyze large datasets, identify patterns, and make data-driven decisions to optimize testing efforts and ensure the quality of software products. This paper explores the intersection of function points, machine learning, and software testing, focusing specifically on projects developed using the Python programming language. Python has gained widespread adoption in the software industry due to its simplicity, readability, and versatility, making it an ideal platform for conducting empirical studies and implementing machine learning algorithms.

This study seeks to investigate whether software projects based on function points can be tested efficiently by applying a machine-learning approach. The objective of this investigation is to investigate and evaluate the ability to predict various machine learning models utilizing real data from various software projects. Additionally, identify the strengths and limitations of using function points as a metric for software testing and explore how machine learning techniques can complement traditional testing methodologies. This work intends to offer important insights into the efficacy of function points-based testing in the context of machine learning through an extensive investigation of empirical data and experimental results The findings of this study might help in the development of more practical and efficient testing methods for software projects by practitioners, researchers, and software engineers.

II. RELATED WORK

In the year 2009, Saxena and Shrivastava conducted a study on the importance of function point analysis in software development using UML modeling [2]. Estimating the software's size was a vital task, and numerous metrics—such as function point analysis, lines of code, and objects—were used for this. A popular method for evaluating system performance was Function Point (FP) analysis. Although FP analysis has been applied to object-oriented software development before, this study investigated its use with the Unified Modeling Language (UML). The paper provided a case study of a Web-based document management system that showed FP analysis using a UML class diagram.

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

In the year 2020, At the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Aljamaan and Alazba [3] presented an empirical evaluation of seven tree-based ensembles for software fault prediction. The study, which included Random Forest, Extra Trees, AdaBoost, Gradient Boosting, Hist Gradient Boosting, XGBoost, and CatBoost, utilized 11 publicly available MDP NASA software defect datasets. Results indicated the superior prediction performance of tree-based bagging ensembles, particularly Random Forest and Extra Trees, over boosting ensembles. Prabha and Shivakumar [4] presented a study at the 4th International Conference on Trends in Electronics and Informatics, focusing on software defect prediction using machine learning techniques. The hybridized approach incorporated PCA, Random Forest, Naïve Bayes, and SVM within the software framework, utilizing five datasets (PC3, MW1, KC1, PC4, and CM1) through the Weka simulation tool. The research analysis evaluated parameters such as confusion, precision, recall, and recognition accuracy, comparing them with existing schemes. Results suggested that the proposed approach could offer more effective solutions for software defect prediction. Morasca and Lavazza [5] conducted a study on the assessment of software defect prediction models using ROC curves and emphasized the importance of setting a threshold (t) on defect proneness models to classify software modules as faulty or non-faulty. The study introduced a new approach and performance metric, the Ratio of Relevant Areas, which considers only parts of the ROC curve where defect proneness models outperform a reference value. The authors addressed shortcomings of existing metrics, derived reference values based on random defect prediction policies, and conducted an empirical study on real-life data to demonstrate the effectiveness of their metric. Cetiner and Sahingoz [6] conducted a comparative analysis of machine learning-based software defect prediction systems. Emphasizing the importance of predicting software defects to enhance software quality and reduce costs in the software development lifecycle, the study compared 10 learning algorithms, including Decision Tree, Naive Bayes, K-Nearest Neighbor, Support Vector Machine, Random Forest, Extra Trees, Adaboost, Gradient Boosting, Bagging, and Multi-Layer Perceptron, using public datasets from the PROMISE warehouse. The experimental results demonstrated that the proposed models achieved proper accuracy levels for software defect prediction, contributing to improved software quality.

In the year 2021, Pan et al. [7] proposed CodeBERT models, including CodeBERT-NT, CodeBERT-PS, CodeBERT-PK, and CodeBERT-PT, for software defect prediction. The empirical study revealed improved prediction performance using pre-trained CodeBERT models, saving time costs. The research also highlighted the effectiveness of sentence-based and keyword-based prediction patterns in enhancing the performance of neural language models for software defect prediction. Wang et al. [8] introduced a Software Defect Prediction (SDP) model based on LASSO–SVM, aiming to address challenges in managing overwhelming software defect reports. The proposed model combines a feature selection method, minimum absolute value compression, and selection, with the support vector machine algorithm. This approach reduces the dimension of the original dataset and eliminates irrelevant data. The optimal value of SVM is determined through parameter optimization using a cross-validation algorithm, followed by SDP accomplished through SVM's nonlinear computing ability. The simulation results demonstrated a higher prediction accuracy of 93.25%, a recall rate of 78.04%, and an f-metric of 72.72% compared to traditional defect prediction models, emphasizing improved accuracy and faster prediction speed.

In the year 2022, Pachouly et al. [9] conducted a systematic literature review on AI-based software defect prediction. The study focused on datasets, data validation, approaches, and tools, revealing shortcomings in standard datasets, such as insufficient features and data validation. The survey emphasized the limited labels in standard datasets and called for more SLRs on Software Defect Prediction. It provided a detailed analysis of defect datasets, validation, detection, prediction approaches, and tools, offering insights for future research. The survey introduced an architecture for constructing a software prediction dataset with features and validation techniques for multi-label classification of software defects. Goyal [10] explored effective software defect prediction using SVM’s. Software defect prediction is crucial for delivering high-quality software products on time. Early detection of error-prone modules in the development phases is essential to prevent potential failures. SVMs are widely used for software defect prediction, but the unequal count of faulty and non-faulty modules in the dataset poses challenges to accuracy. The study proposed a novel filtering technique (FILTER) to enhance the performance of SVM-based defect prediction models. SVM classifiers (linear, polynomial, and radial basis function) were designed with the proposed filtering technique, showing improvements of 16.73%, 16.80%, and 7.65% in accuracy, AUC, and F-measure, respectively. Jorayeva et al. [11] conducted a systematic literature review on machine learning-based software defect prediction for mobile applications. The study addressed nine research questions, revealing a focus on Android applications (48%), with 92% utilizing supervised machine learning. Object-oriented metrics were predominantly preferred, and the top five machine learning algorithms included Naïve Bayes, Support Vector Machines, Logistic Regression, Artificial Neural Networks, and Decision Trees. Deep learning algorithms like Long Short-Term Memory (LSTM), Deep Belief Networks (DBN), and Deep Neural Networks (DNN) were applied in only a few studies. This SLR provides a comprehensive overview of software defect prediction in the context of mobile applications, guiding future research and aiding practitioners in the field.

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

In the year 2023, Khalid et al. [12] conducted a study on software defect prediction using machine learning techniques. The research aimed to enhance model performance in terms of accuracy and precision compared to prior studies. K-means clustering was employed for class label categorization, and classification models were applied to selected features. Particle Swarm Optimization was used to optimize machine learning models. The evaluation metrics included precision, accuracy, recall, f-measure, performance error metrics, and a confusion matrix. Results showed that SVM and optimized SVM models outperformed others, achieving the highest accuracy of 99% and 99.80%, respectively. The study demonstrated improved accuracy compared to previous research. Kumar et al. [13] conducted a study on the generation of test cases for the identification of crimes against women through HLR (Home Location Register) and VLR (Visitor Location Register). The research aimed to address the increasing crimes against women by establishing a criminal scene through mathematical techniques, and matching home and visitor locations to identify crimes. The study introduced a Unified Model using the Unified Modeling Language, which was validated through test cases implemented in the Python programming language. Kumar and Saxena [14] proposed a Predictive Test Prioritization (PTP) approach using machine learning classifiers on historical test data. Their method improved test case prioritization in CI, enhancing efficiency based on F1-score, recall, accuracy, and precision.

III. METHODOLOGY

In the proposed work, a system model is designed and given below in figure 1.

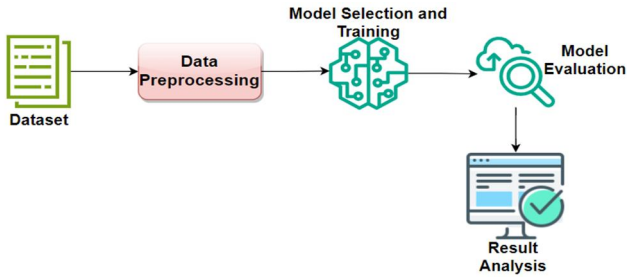

FIGURE 1. SYSTEM MODEL FOR INVESTIGATING DEFECT PREDICTION

1. Dataset

The selected dataset, "Class-level data for KC1," is derived from the scholarly work of A. Gunes Koru and is timestamped February 21, 2005[1]. This dataset has previously served as a basis for examining defect prediction within the realm of software engineering. It encompasses class-level attributes, which include metrics obtained from function points and lines of code, thereby offering valuable insights into the characteristics of software projects. Significant characteristics within the dataset consist of proportions of public and protected data, inter-object coupling, class depth, cohesion metrics, and polymorphism, among others. The dependent variable, "DL" (Defect Level), is dichotomous, signifying either the existence (_TRUE) or nonexistence (FALSE) of defects within a class. The attributes and dependent variables associated with this dataset render it particularly suitable for investigating defect prediction models, with a specific emphasis on the efficacy of function points in foreseeing defects within software classes.

2. Data Preparation: Preprocessing, Feature Engineering, and Data Splitting

The process of data preparation, as delineated in the accompanying code, encompasses several critical stages to adequately prepare the dataset for the training of machine learning models. Initially, preprocessing addresses the issue of missing values within numerical features through the application of mean imputation utilizing the SimpleImputer class from the scikit-learn library. This approach guarantees that the dataset is complete and primed for subsequent analytical endeavors. Although there exists a commentary alluding to the possible necessity for the encoding of categorical variables, the precise encoding methodology is not articulated within the code. The target variable 'DL,' which signifies class defectiveness, is subjected to encoding into a numerical representation via LabelEncoder, thereby facilitating binary classification wherein '_TRUE' is designated as 1 and 'FALSE' as 0. Subsequent to the preprocessing phase, feature engineering is executed to convert method-level features into class-level features. This conversion entails the aggregation of features such as LOC_BLANK and BRANCH_COUNT by calculating the minimum, maximum, sum, and average values across all methods associated with a particular class. By acquiring a more extensive array of information regarding the behavioral patterns of each class, this feature engineering phase significantly enhances the dataset for subsequent model training. Ultimately, the dataset undergoes stratification and division into training and testing subsets utilizing the train_test_split function. This methodical partitioning guarantees that both subsets maintain a proportional distribution of target classes, with 80% of the dataset designated for training and 20% for testing. Such a methodology fosters impartial model assessment and contributes to the overall resilience of the machine learning model.

The data preparation procedure entails the resolution of absent values, the potential encoding of categorical variables, feature engineering for improved representation, and the implementation of stratified data division. These actions collectively create a meticulously prepared dataset, thereby establishing a robust foundation for the effective training and evaluation of machine learning models within the domain of software class defect prediction.

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

3. Model Selection and Training

In this phase of the methodology, we embark on the meticulous selection and training of machine learning models to facilitate defect prediction in software projects. Drawing from the versatile scikit-learn library in Python, a diverse ensemble of machine-learning models tailored for binary classification has been curated. The ensemble includes the Random Forest classifier, Logistic Regression, Decision Tree, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Each model has been chosen for its distinct characteristics, and the intention is to comprehensively evaluate its efficacy within the context of the dataset. A distinctive aspect of this approach involves the utilization of a Random Forest ensemble classifier. Hyperparameter tuning was conducted through GridSearchCV, systematically exploring various hyperparameter combinations, including the number of trees (n_estimators), maximum tree depth (max_depth), minimum samples required for node splitting (min_samples_split), and minimum samples required for leaf nodes (min_samples_leaf). The objective is to identify the optimal hyperparameter configuration that maximizes the predictive capabilities of the Random Forest model. The training process commenced by fitting each selected model to the training dataset (X_train_scaled, y_train). During this phase, the models assimilated the underlying patterns and relationships between the input features and the target variable, representing the defect level (DL). Notably, the training data underwent scaling using StandardScaler to normalize numerical features, ensuring uniformity in feature scales.

A unique inclusion in the ensemble is the Stacking Classifier, a meta-learner that combines the predictions of multiple base models. The base models include the Random Forest, Logistic Regression, Decision Tree, SVM, and KNN. The Stacking Classifier employs a neural network, specifically an MLPClassifier, as the final estimator, providing enhanced complexity and adaptability to the ensemble. Following the training phase, model evaluation transpired on the testing set (X_test_scaled, y_test). Performance metrics, encompassing accuracy, precision, recall, and F1 score, were computed to gauge the models' effectiveness in correctly classifying defective and non-defective software classes. This evaluation process aims to offer a nuanced understanding of each model's strengths and weaknesses, guiding the identification of the most suitable model for defect prediction based on function points. Insights derived from this evaluation, including the unique contributions of the Stacking Classifier, will be pivotal in shaping subsequent analysis and decision-making regarding the optimal model for software defect prediction.

4. Model Evaluation

During the model evaluation phase, the effectiveness of the chosen machine learning models in predicting the defectiveness of software classes, based on features derived from function points, is critically assessed. This evaluation unfolds after the models undergo training on the specified training set, with a primary focus on evaluating their performance on the designated testing set. This section delineates the steps integral to model evaluation, delineating the metrics utilized and the subsequent analysis of the outcomes. To gauge the predictive capabilities of the models accurately, a suite of standard classification metrics is employed. These metrics encompass accuracy, measuring the proportion of correctly classified instances among the total instances; precision, which denotes the ratio of true positive predictions to the total positive predictions, indicative of the model's adeptness in avoiding false positives; recall, represented by the ratio of true positive predictions to the total actual positives, reflecting the model's efficacy in capturing all positive instances; and the F1 Score, serving as the harmonic mean of precision and recall, thereby providing a balanced assessment of a model's overall performance.

In conducting the model-specific evaluation, each of the selected models—namely, the Random Forest, Logistic Regression, Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and the Stacking Classifier with a neural network final estimator—undergoes a standardized evaluation procedure. This process entails training and subsequently evaluating the models based on their default configurations, except the Random Forest model, which undergoes hyperparameter tuning using GridSearchCV to optimize its parameters such as the number of estimators, maximum depth, minimum samples split, and minimum samples leaf. This meticulous evaluation approach facilitates a comprehensive assessment of each model's predictive prowess, aiding in informed decision-making regarding their suitability for predicting software class defectiveness.

5. Result Analysis

The computed metrics (accuracy, precision, recall, F1 score) for each model are analyzed to discern patterns and understand the strengths and weaknesses of the models. Comparative insights into their performance, particularly in the context of function points and class-level data, are drawn. Visualizations, such as tables and graphs, are utilized to present the evaluation results comprehensively, aiding in the interpretation of the model's predictive capabilities.

IV. RESULTS AND DISCUSSION

The culmination of the model evaluation phase yields valuable insights into the performance of various machine learning models in predicting the defectiveness of software classes based on function points and class-level data. The results, presented through a detailed analysis of key

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

metrics, pave the way for informed discussions and nuanced interpretations.

1. Accuracy Analysis

Accuracy serves as a pivotal metric in assessing the efficacy of machine learning models for predicting software class defectiveness based on function points and class-level data. The accuracy scores for different models provide valuable insights into their overall performance, aiding in the selection of optimal models for defect prediction tasks. The following table gives information about the accuracy of the model. A similar representation is shown below in the figure 1.

Table 1. Comparative Analysis of Models Accuracy

Model	Accuracy
Random Forest	0.7931
Logistic Regression	0.7931
Decision Tree	0.7241
SVC	0.7586
KNN	0.8276
Stacking Classifier with NN	0.8621

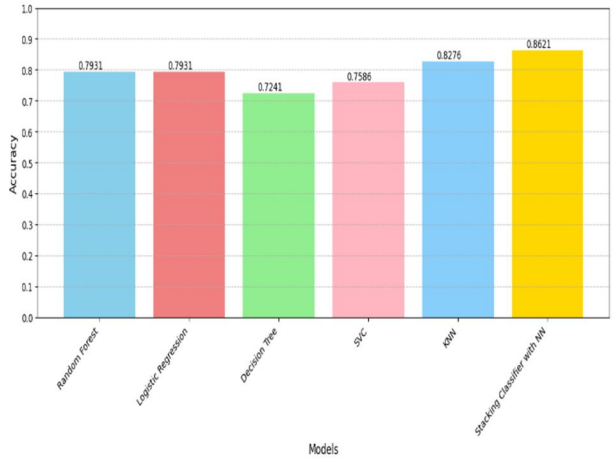

Figure 2. Representation of Accuracy of Models

The Random Forest and Logistic Regression models demonstrate analogous accuracy, each attaining a score of 0.7931. This indicates that approximately 79.31% of the predictions generated by these models correspond with the actual defectiveness labels present in the testing dataset. These models display uniform and competitive performance within the ensemble framework. Conversely, the Decision Tree model registers a marginally diminished accuracy of 0.7241. This implies that the Decision Tree model may encounter difficulties in encapsulating the intrinsic complexity of patterns within the dataset, leading to a reduced overall predictive accuracy. The Support Vector Machine (SVC) achieves an accuracy of 0.7586, situating it competitively yet slightly inferior to Random Forest and Logistic Regression. This suggests that the SVC model may exhibit variability in its predictive proficiency relative to other models. The K-Nearest Neighbors (KNN) model emerges as a high achiever with an accuracy of 0.8276, surpassing other models, including Random Forest and Logistic Regression. This indicates that KNN excels in the precise classification of instances within the testing dataset. Notably, the Stacking Classifier employing a Neural Network final estimator reaches the pinnacle accuracy of 0.8621 among all models. This underscores the efficacy of the ensemble methodology, harnessing a combination of foundational models to attain superior predictive accuracy concerning defectiveness in software classes. In pragmatic terms, these accuracy metrics provide valuable insights to software practitioners and decision-makers in the selection of models for defect prediction endeavors. The findings emphasize the effectiveness of ensemble methodologies, particularly stacking, in augmenting predictive capabilities. Nonetheless, it is imperative to complement accuracy evaluation with additional metrics for a holistic appraisal of model performance.

2. Precision Evaluation

Precision, an essential metric for assessing the efficacy of machine learning models, plays a pivotal role in quantifying the accuracy of positive predictions, especially when forecasting software class defectiveness utilizing function points and class-level data. The precision scores across diverse models illuminate their capacity to mitigate false positives while ensuring the accuracy of positive predictions. Commencing with the Random Forest model, it attains a precision score of 0.5833, signifying that approximately 58.33% of the instances classified as defective are indeed true positives. This model exhibits a moderate degree of precision, adeptly balancing accurate defect predictions while circumventing an excessive number of false positives. Logistic Regression closely parallels this performance, achieving a precision score of 0.6000, thereby reinforcing its reliable capacity for accurate positive predictions. The Decision Tree model demonstrates a precision score of 0.5000, indicating that fifty percent of the positive predictions rendered by the model are genuine positives. Although the precision remains moderate, the model may face difficulties in differentiating between true positives and false positives. The Support Vector Machine (SVC) records a precision score of 0.5385, which is marginally lower than the precision scores of both Random Forest and Logistic Regression. K-Nearest Neighbors (KNN) distinguishes itself with a precision score of 0.6667, indicating that approximately 66.67% of the positive predictions made by KNN are indeed true positives. This reflects a superior level of precision, indicative of effective identification of defective software classes. The Stacking Classifier, utilizing a Neural Network as the final estimator, achieves the utmost precision score of 0.7000, illustrating that 70.00% of the positive predictions generated by the stacking model are accurate, thereby underscoring its effectiveness in reducing false positives.Precision scores

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

offer valuable insights into each model's ability to make accurate positive predictions. The Stacking Classifier with a Neural Network emerges as the top performer in terms of precision, followed by KNN, Logistic Regression, SVC, Random Forest, and Decision Tree. These precision scores provide a nuanced perspective for selecting models in defect prediction tasks, underscoring the importance of precision alongside other evaluation metrics for a comprehensive assessment of model performance. Precision assessment across models is represented below:

Table 2. Precision Assessment Across Models

Model	Precision
Random Forest	0.5833
Logistic Regression	0.6000
Decision Tree	0.5000
SVC	0.5385
KNN	0.6667
Stacking Classifier with NN	0.7000

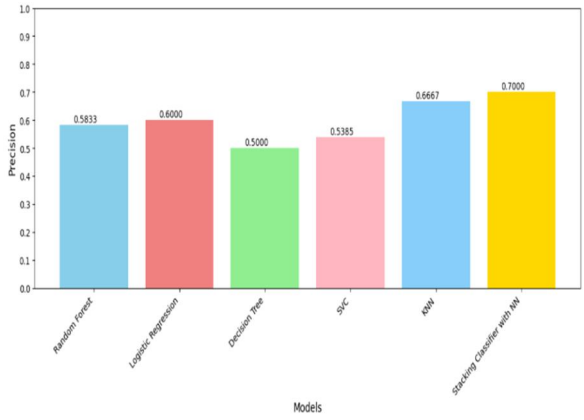

Figure 3. Representation of Precision Assessment Across Models

3. Recall Evaluation

Recall, a fundamental metric for classification, evaluates the proficiency of machine learning models in identifying all positive instances, particularly in the context of predicting software class defectiveness based on function points and class-level data. The recall metrics for each respective model illuminate their efficacy in recognizing and accurately categorizing defective instances. The Random Forest model excels in recall, attaining a score of 0.8750. This indicates that approximately 87.50% of genuine defective instances are accurately recognized by the Random Forest model. The elevated recall score signifies the model's robustness in capturing the predominant share of positive cases. Logistic Regression achieves a recall score of 0.7500, which denotes that 75.00% of genuine defective instances are accurately identified by the model. Although this score is marginally lower than that of Random Forest, the model continues to exhibit a commendable capacity for identifying positive instances. The Decision Tree model displays a recall score of 0.6250, suggesting that 62.50% of genuine defective instances are accurately identified. Although this performance is moderate, the model may encounter limitations in capturing a substantial share of positive cases. The Support Vector Machine (SVC) attains a recall score of 0.8750, reflecting the recall capability of Random Forest. This signifies that the SVC model effectively identifies a considerable proportion of genuine defective instances. K-Nearest Neighbors (KNN) achieves a recall score of 0.7500, which is consistent with the recall performance of Logistic Regression. This suggests that 75.00% of genuine defective instances are accurately recognized by the KNN model. The Stacking Classifier employing a Neural Network secures a recall score of 0.8750, paralleling the recall performance of both Random Forest and SVC. This underscores the effectiveness of the stacking model in capturing the majority of genuine defective instances. Recall scores furnish valuable insights into each model's aptitude for accurately identifying positive cases. The Stacking Classifier utilizing a Neural Network, Random Forest, and SVC are distinguished as the foremost performers in terms of recall, underscoring their effectiveness in detecting genuine defective instances.These recall scores contribute valuable information for selecting models in defect prediction tasks, complementing other evaluation metrics for a comprehensive assessment of model performance. Recall evaluation across models is given below:

Table 3 Models Recall Evaluation

Model	Recall
Random Forest	0.8750
Logistic Regression	0.7500
Decision Tree	0.6250
SVC	0.8750
KNN	0.7500
Stacking Classifier with NN	0.8750

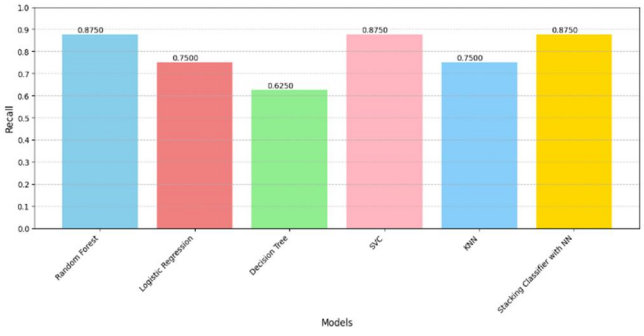

Figure 4. Representation of Recall Evaluation Across Models

4. F1 Score Evaluation

The F1 Score, an essential metric that reconciles precision and recall, provides a thorough evaluation of the

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

performance of machine learning models in the context of predicting software class defectiveness, utilizing function points and class-level data. The F1 Score integrates both false positives and false negatives, thereby offering a sophisticated understanding of a model's overall efficacy. The Random Forest model attains an F1 Score of 0.7000, illustrating a well-balanced performance in the reduction of false positives and false negatives. This score denotes a harmonious equilibrium between precision and recall, highlighting the model's capacity to deliver both accurate and comprehensive predictions. Logistic Regression secures an F1 Score of 0.6667, reflecting a commendable balance in terms of precision and recall. This score encapsulates the model's proficiency in generating accurate positive predictions while adeptly identifying actual defective instances. The Decision Tree model presents an F1 Score of 0.5556, indicating a moderate equilibrium between precision and recall. The model may face obstacles in effectively minimizing both false positives and false negatives, which results in a diminished overall F1 Score. The Support Vector Machine (SVC) achieves an F1 Score of 0.6667, corresponding with the performance of Logistic Regression. This score indicates a well-balanced performance, achieving a compromise between precision and recall. K-Nearest Neighbors (KNN) distinguishes itself with an F1 Score of 0.7059, signifying a superior balance between precision and recall. This suggests the model's effectiveness in generating accurate positive predictions while successfully capturing a substantial proportion of actual defective instances. The Stacking Classifier utilizing a Neural Network achieves the highest F1 Score of 0.7778, underscoring exceptional performance in reconciling precision and recall. This score highlights the model's capability to provide both precise and comprehensive predictions, establishing it as a leading performer in terms of F1 Score. F1 Scores furnish a comprehensive perspective on the efficacy of models, taking into account the equilibrium between precision and recall. The Stacking Classifier integrated with a Neural Network is identified as the foremost performer, succeeded by KNN, Random Forest, Logistic Regression, SVC, and Decision Tree. These metrics yield significant insights pertinent to the selection of models in defect prediction endeavors, guaranteeing a judicious consideration of both false positives and false negatives for a thorough assessment of model efficacy. A comparative analysis of F1 scores among the models is presented below:

Table 4. F1 Score Comparison Among Models

Model	F1 Score
Random Forest	0.7000
Logistic Regression	0.6667
Decision Tree	0.5556
SVC	0.6667
KNN	0.7059
Stacking Classifier with NN	0.7778

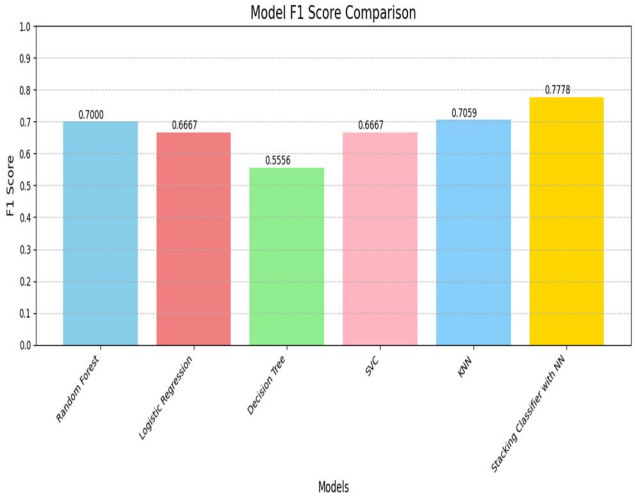

Figure 5. Representation of F1 Score Comparison Among Models

V. CONCLUSIONS

The present study is based on testing of software projects based on function points through a machine-learning approach that yielded promising results, with the Stacking Classifier with a Neural Network standing out as a top performer. This underscores the potential of machine learning in enhancing defect prediction accuracy and guiding quality assurance activities in software development. Looking ahead, future research should explore additional machine-learning algorithms, ensemble techniques, and diverse datasets to further refine and generalize the proposed approach. Incorporating advanced feature engineering, selection methods, and deep learning architectures could provide a deeper understanding of complex relationships within software projects. Collaborative efforts between academia and industry practitioners will be crucial for collecting richer datasets and developing context-specific models.

The intersection of machine learning and software testing offers exciting possibilities for improving software quality and reliability. Our study contributes valuable insights to this dynamic field, and ongoing efforts will continue to shape the future of effective testing strategies in software engineering.

REFERENCES

[1] http://promise.site.uottawa.ca/SERepository/datasets/kc1-class-level-defectiveornot.arff.

[2] Saxena, V., & Shrivastava, M. (2009). Performance of function point analysis through UML modeling. *ACM SIGSOFT software engineering notes*, *34*(2), 1-4, DOI: https://doi.org/10.1145/1507195.1507214.

[3] Aljamaan, H., & Alazba, A. (2020, November). Software defect prediction using tree-based ensembles. In *Proceedings of the 16th ACM*

Volume: 16 Issue: 05 Pages: 6583-6590 (2025) ISSN: 0975-0290

international conference on predictive models and data analytics in software engineering (pp. 1-10), DOI: https://doi.org/10.1145/3416508.3417114.

[4] Prabha, C. L., & Shivakumar, N. (2020, June). Software defect prediction using machine learning techniques. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)* (pp. 728-733). IEEE, DOI: https://doi.org/10.1109/ICOEI48184.2020.9142909.

[5] Morasca, S., & Lavazza, L. (2020). On the assessment of software defect prediction models via ROC curves. *Empirical Software Engineering*, *25*, 3977-4019, DOI: https://doi.org/10.1007/s10664-020-09861-4.

[6] Cetiner, M., & Sahingoz, O. K. (2020, July). A comparative analysis for machine learning based software defect prediction systems. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE, DOI: https://doi.org/10.1109/ICCCNT49239.2020.9225352.

[7] Pan, C., Lu, M., & Xu, B. (2021). An empirical study on software defect prediction using codebert model. *Applied Sciences*, *11*(11), 4793, DOI: https://doi.org/10.3390/app11114793.

[8] Wang, K., Liu, L., Yuan, C., & Wang, Z. (2021). Software defect prediction model based on LASSO–SVM. *Neural Computing and Applications*, *33*, 8249-8259, DOI: https://doi.org/10.1007/s00521-020-04960-1.

[9] Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., & Abraham, A. (2022). A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. *Engineering Applications of Artificial Intelligence*, *111*, 104773, DOI: https://doi.org/10.1016/j.engappai.2022.104773.

[10] Goyal, S. (2022). Effective software defect prediction using support vector machines (SVMs). *International Journal of System Assurance Engineering and Management*, *13*(2), 681-696, DOI: https://doi.org/10.1007/s13198-021-01326-1.

[11] Jorayeva, M., Akbulut, A., Catal, C., & Mishra, A. (2022). Machine learning-based software defect prediction for mobile applications: A systematic literature review. *Sensors*, *22*(7), 2551, DOI: https://doi.org/10.3390/s22072551.

[12] Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software Defect Prediction Analysis Using Machine Learning Techniques. *Sustainability*, *15*(6), 5517, DOI: https://doi.org/10.3390/su15065517.

[13] Kumar, H., Shukla, R., Gautam, P. K., Tiwari, M., & Saxena, V. (2023). Generation of Test Cases for Identification of Crime against Women through HLR and VLR. *Journal of Advances in Mathematics and Computer Science*, *38*(12), 50-59, DOI: https://doi.org/10.9734/jamcs/2023/v38i121857.

[14] Kumar, H., & Saxena, V. (2024). Machine Learning for Test Case Prioritization in Continuous Integration: A Comprehensive Analysis. *Int. J. Advanced Networking and Applications*, *15*(06), 6218-6228.