iJETRM
International Journal of Engineering Technology Research & Management
(IJETRM)
https://ijetrm.com/

INTELLIGUARD: ENHANCED SOFTWARE DEFECT FORECASTING WITH ML

DIVIDEVARA SAI TEJA [1],
G. PRAVEEN BABU[2]
[1]Post Graduate Student, MCA, Department of Information Technology, Jawaharlal Nehru Technological University, Hyderabad, India
[2] Associate Professor, Department of Information Technology, Jawaharlal Nehru Technological University, Hyderabad, India

ABSTRACT
The rising complexity of modern software has made defect management a persistent challenge, as traditional testing and manual reviews often miss fault-prone modules. These overlooked defects increase development costs, reduce system reliability, and can even lead to project failure. To address this, IntelliGuard introduces a machine-learning-driven solution that predicts defective modules before they cause downstream issues. By leveraging historical defect data and extracting meaningful metrics, the system enables teams to focus on high-risk components and optimize testing efforts. At its core, IntelliGuard uses a structured ML pipeline with preprocessing, feature selection, and a Random Forest classifier designed to handle imbalanced datasets through weighted learning.IntelliGuard is deployed as an interactive Streamlit application that supports both single-module analysis and bulk prediction via CSV uploads. The app also offers real-time visualizations such as performance metrics, confusion matrix plots, precision–recall curves, and feature-importance charts. Extensive experiments show that the system effectively balances precision and recall, ensuring reliable detection of both defective and clean modules.

Keywords:
Software Defect Prediction, Machine Learning, Quality Assurance, RandomForestClassifier, Static Code Metrics, Data-Driven Testing, Proactive Analysis, Streamlit, Scikit-learn.

I. INTRODUCTION
Software now powers nearly every modern industry, making reliability more critical than ever. Defects in code threaten this reliability, but traditional quality assurance struggles to keep up with the scale and complexity of current software systems. This creates a need for intelligent, data-driven approaches that can spot defect-prone modules early. Machine learning offers a strong solution by analyzing historical defect patterns and code metrics to predict which components are likely to fail, helping teams focus their testing efforts where it matters most. However, practical challenges such as class imbalance, redundant features, and poor interpretability still limit adoption. IntelliGuard addresses these gaps through a Random Forest–based pipeline enhanced with feature selection, normalization, and class-weight balancing to ensure more reliable predictions. The model is deployed in a Streamlit dashboard that supports single-module and bulk analysis while providing performance metrics, confusion matrices, precision–recall curves, and feature importance visualizations.

Objective:
The primary aim of IntelliGuard is to design and implement a predictive framework that enhances the accuracy and practicality of software defect forecasting using machine learning.

II. Literature Survey
A literature survey on software defect prediction shows how the field has evolved from basic manual inspections and traditional metrics like LOC and cyclomatic complexity to more sophisticated data-driven techniques. Early metric-based methods offered useful insights but lacked strong predictive capability and often failed to generalize across different projects. As computing advanced, researchers adopted statistical and machine learning models—decision trees, logistic regression, SVMs, and neural networks—which improved accuracy but struggled with imbalanced datasets and limited interpretability. Ensemble models such as Random Forests and boosting techniques later emerged as more robust solutions, especially for noisy or complex datasets. Studies also highlight two consistent needs~~: interpretability, because many ML models act as black boxes, and feature selection, to determine which~~ metrics most strongly correlate with defects.

iJETRM
International Journal of Engineering Technology Research & Management
(IJETRM)
https://ijetrm.com/

III. Methodology

3.1 Proposed System

To overcome the limitations of traditional software defect management practices, this project introduces IntelliGuard, an enhanced software defect forecasting framework that integrates machine learning with an interactive, user-friendly platform. The system shifts defect management from a reactive process to a proactive strategy by leveraging historical datasets, predictive modelling, and intelligent visualization techniques. By forecasting defect-prone modules before deployment, IntelliGuard enables early intervention, reduces defect leakage, and enhances the overall reliability of software products

3.2 Dataset Description

Source: The project uses the NASA KC1 dataset, a standard benchmark from the PROMISE repository.
Content: The model was trained on a dataset of 2,109 software modules, where each module is described by 21 static code metrics (e.g., McCabe's Complexity, Halstead's Volume).
Origin: The data originates from a real-world, mission-critical software project written in C++.
Target Variable: Each module in the dataset is labelled with a ground truth outcome: **defective (1)** or **clean (0)**.
Key Challenge: The dataset is naturally imbalanced, containing significantly more clean modules than defective ones. This was a critical factor that influenced our model training strategy to ensure it could effectively identify the rare but important defective cases.

3.3 System Architecture

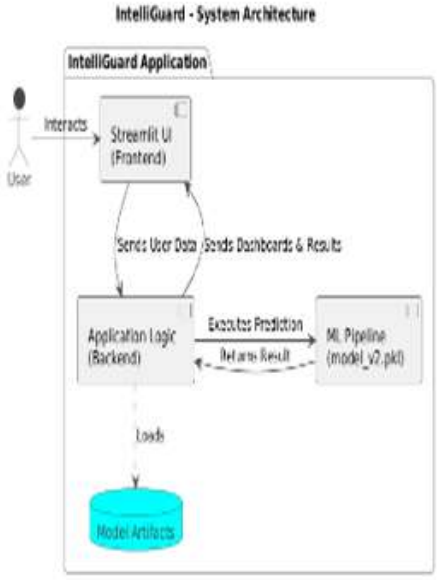

Figure.1: System Architecture Diagram.

The architecture of IntelliGuard is designed as a modular, two-tier system to effectively separate the user interface from the core machine learning logic. The first tier is the Frontend Presentation Layer, a web application built entirely with the Streamlit framework, which is responsible for capturing all user input and displaying the final analysis. The second tier is the Backend Logic Layer, a Python script that uses the Pandas library for data handling and the Scikit-learn library to execute the pre-trained machine learning pipeline. This separation of concerns ensures that the user interface is decoupled from the data processing and prediction engine, making the system more robust, scalable, and easier to maintain.

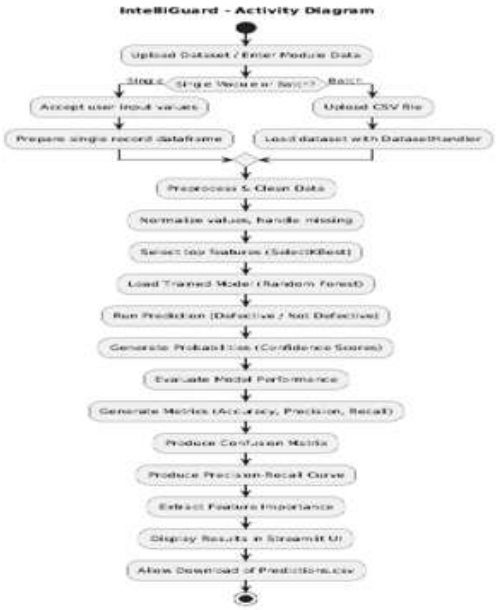

Figure.2: Activity Diagram.

The process begins when the user launches the application and is immediately presented with a decision point: to analyze a single module or analyze from a CSV file. If the single module path is chosen, the flow is linear, proceeding from data entry to prediction and result display. If the CSV path is chosen, the workflow includes steps for uploading the file, optionally editing the data in the interactive grid, running the prediction, and finally, viewing the comprehensive results table and performance dashboards. This diagram effectively visualizes the different paths a user can take and the sequence of actions required to achieve a result within the IntelliGuard application

iJETRM
International Journal of Engineering Technology Research & Management
(IJETRM)
https://ijetrm.com/

3.4 Methodology (ML Pipeline)

The machine learning pipeline was executed as follows:

1. **Purpose & Benefit:** The Pipeline chains all processing steps into a single, cohesive object. This is a critical best practice that ensures the exact same transformations are applied to both the training data and any new data, which guarantees consistency and prevents common errors like data leakage.
2. **StandardScaler (Data Scaling):** The first step in the pipeline standardizes all 21 numeric features. This transformation prevents metrics with naturally large value ranges from disproportionately influencing the model's predictions.
3. **SelectKBest (Feature Selection):** The second step automatically analyses the scaled features and selects the 15 most statistically significant ones. This simplifies the model by removing noisy or irrelevant metrics, improving robustness and efficiency.
4. **RandomForestClassifier (Prediction):** The final stage takes the cleaned and selected features and uses the powerful Random Forest algorithm to make the "Defective" or "Not Defective" classification. This model was specifically trained to handle the dataset's class imbalance, making it highly effective at its predictive task.

IV. Experiment and Analysis

The experiment involved training a balanced RandomForest model on 80% of the KC1 dataset and then evaluating it on a hidden 20% test set. Analysis of the results confirmed the model is a high-precision tool, achieving 99.4% precision while successfully identifying over 80% of actual software defects.

4.1 Key Features

- Dual Analysis Modes: Provides flexibility with both a real-time prediction for single modules and a batch processing capability for large datasets via CSV upload.
- Interactive "What-If" Analysis: Features an editable data grid that allows users to modify software metrics and instantly re-run predictions to simulate the impact of code refactoring.
- Performance Evaluation Dashboard**:** Automatically generates a visual dashboard with key metrics (Accuracy, Precision, Recall), a Confusion Matrix, and a Precision-Recall Curve to provide a deep, quantitative assessment of the model's performance.
- Model Interpretation Dashboard: Includes a Feature Importance chart that explains the model's predictions by ranking which software metrics are the most influential, turning the model from a "black box" into a transparent and actionable tool.

4.2 Experiment Analysis and Results

The final, trained model was rigorously evaluated on a hidden 20% test set to provide a realistic and unbiased assessment of its performance. The system demonstrated excellent predictive capabilities, achieving an overall accuracy of 93.70%. Critically, it proved to be a high-precision tool with a 99.4% precision rate on its defect predictions, ensuring that its alerts are highly trustworthy and generate minimal false alarms. Furthermore, the model successfully identified over 80% of all actual defects, confirming its effectiveness in risk detection. The interpretation dashboard revealed that the model's decisions were most heavily influenced by metrics related to code size and complexity, providing a clear and actionable insight.

iJETRM
International Journal of Engineering Technology Research & Management
(IJETRM)
https://ijetrm.com/

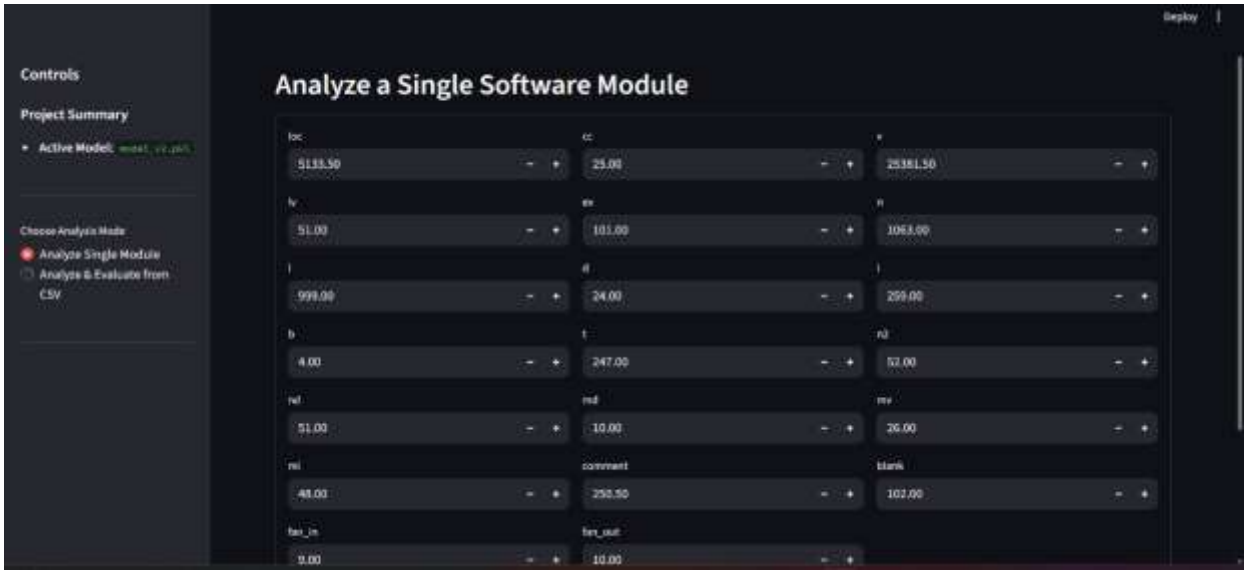

Figure 3: The main interface is organized into two primary sections: a sidebar for global controls and a main panel for analysis and results.

Figure 4: The second core functionality of the IntelliGuard system is its "Analyze & Evaluate from CSV" mode. This feature is designed for processing large datasets and enabling powerful, interactive "what-if" analysis.

iJETRM
International Journal of Engineering Technology Research & Management
(IJETRM)
https://ijetrm.com/

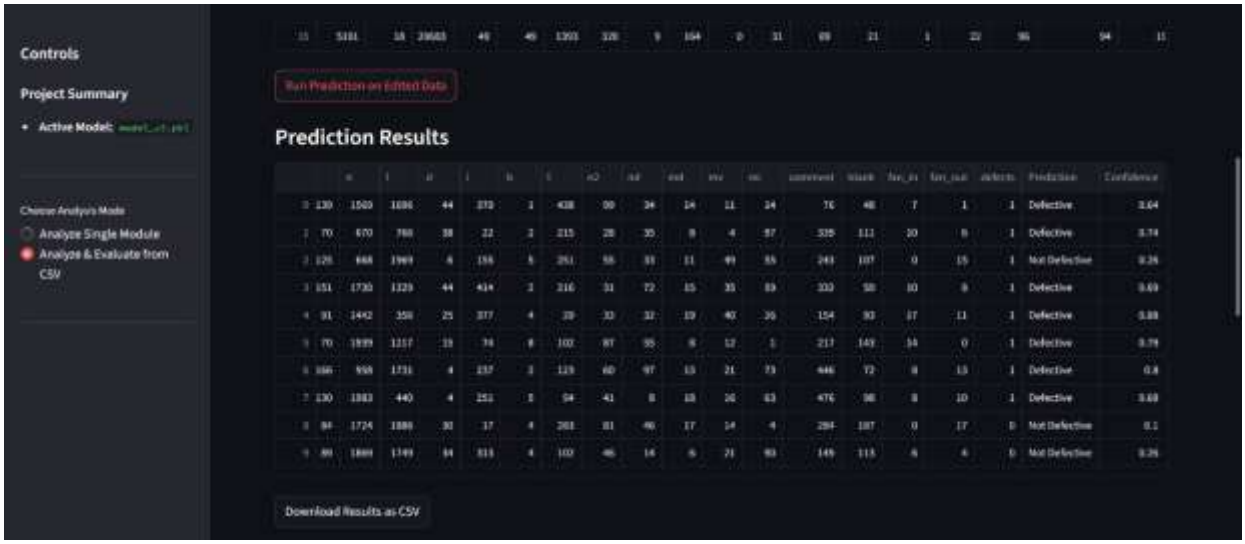

Figure 5: After a user uploads a CSV file and initiates the analysis, the IntelliGuard system processes the data and presents its primary output: a comprehensive and integrated results table. This table is designed to be easily interpretable, combining the original input data with the model's predictive insights.

Figure 6: The dashboard begins with a high-level summary of the model's performance through three critical metrics.

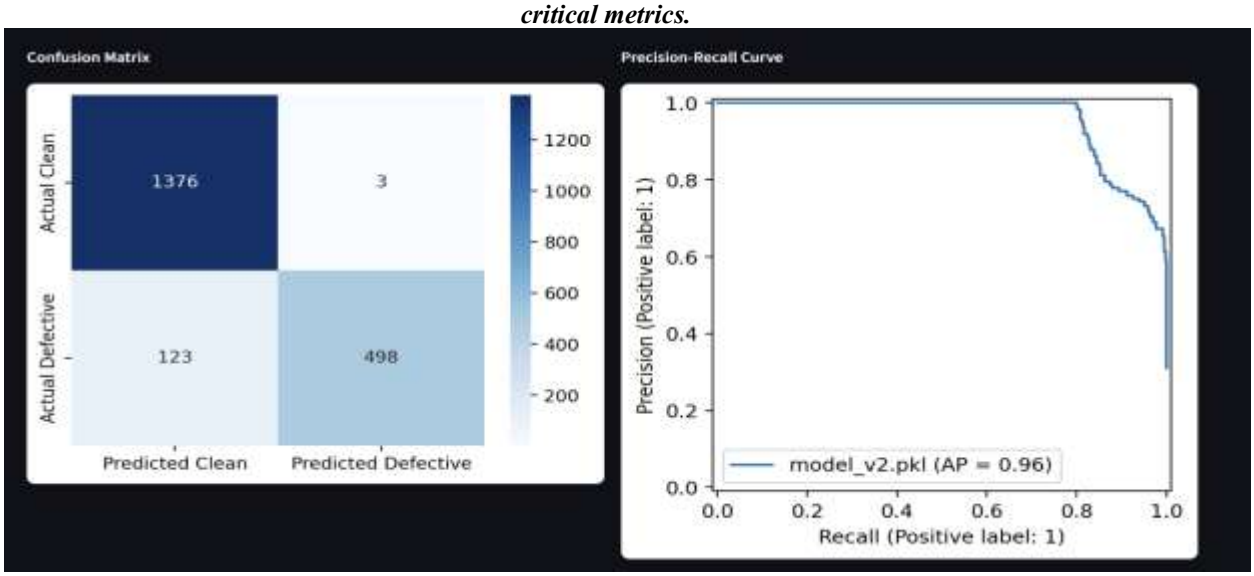

Figure 7: To provide deeper insight beyond the KPIs, the dashboard presents two standard visualizations for classification model analysis.

iJETRM
International Journal of Engineering Technology Research & Management
(IJETRM)
https://ijetrm.com/

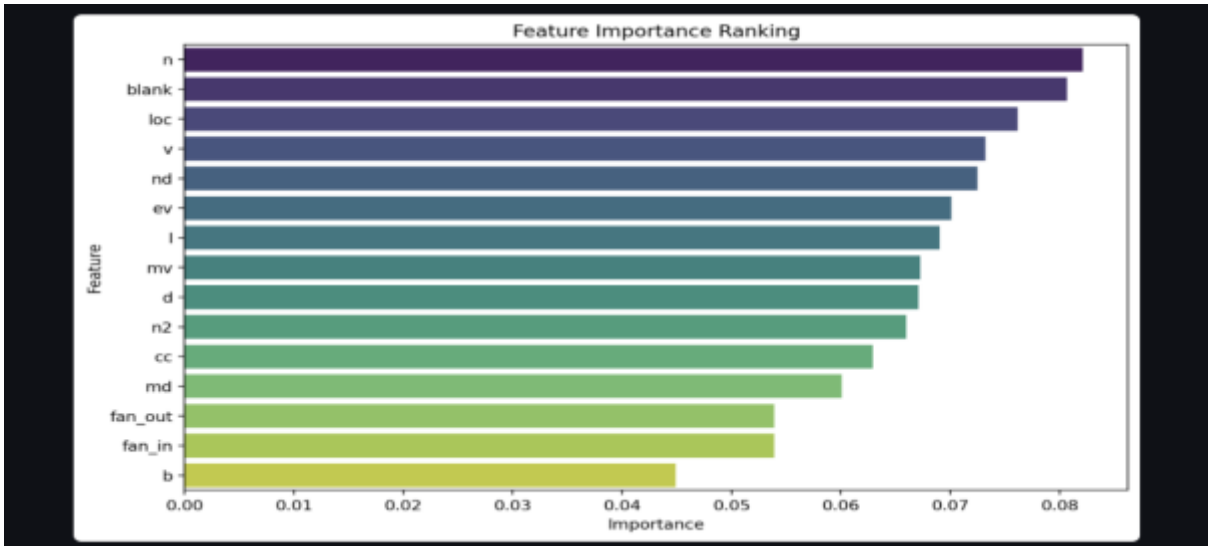

Figure 8: This dashboard visualizes which software metrics the RandomForestClassifier model found most influential when distinguishing between defective and non-defective modules.

V. Limitations and Future Scope

While IntelliGuard serves as a successful proof-of-concept, its primary limitation is its dependency on a single dataset, the NASA KC1 project, which may not generalize perfectly to software written in other languages or in different domains. The model's predictions are also based solely on static code metrics and do not incorporate dynamic, run-time behaviors or process-related metrics like code churn, which are known to influence defect rates. Furthermore, the system's "what-if" analysis is a simulation and cannot guarantee that a refactoring effort will eliminate all bugs. The tool is designed to guide and prioritize human effort, not to replace traditional QA testing.

VI. Conclusion

IntelliGuard successfully demonstrates that a machine learning system can serve as a powerful, proactive tool in software quality assurance. By integrating a high-precision RandomForestClassifier into an interactive web application, this project bridges the gap between theoretical models and practical, usable tools. The system not only predicts defects with high reliability but also provides actionable insights, proving that code size and complexity are the most significant drivers of risk. Ultimately, IntelliGuard serves as a robust proof-of-concept for a data-driven strategy that enables development teams to focus their resources more efficiently and build more reliable software.

VII. References

1. Breiman, L. (2001). "Random Forests." Machine Learning, 45(1), 5-32.
2. Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling. Springer
3. Harris, C. R., et al. (2020). "Array programming with NumPy." Nature, 585(7825), 357-362.
4. McKinney, W. (2010). "Data Structures for Statistical Computing in Python." Proceedings of the 9th Python in Science Conference, 445, 51-56.