



Enhancing Agile Defect Prediction with Optimized Machine Learning and Feature Selection

Faiq Dhimas Wicaksono^{1*}, Daniel Siahaan²

¹Master Program of Technology Management, Interdisciplinary School of Management and Technology, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

²Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

¹6032231111@student.its.ac.id, ²daniel@if.its.ac.id

Abstract

In Agile software development, efficient defect prediction is crucial because of the rapid and iterative nature of the delivery. Conventional methods that rely on source code or commit logs often fail to capture the critical contextual signals necessary for early bug detection. This study proposes a hybrid machine learning framework that leverages enriched contextual features from Jira issue tickets and combines them with optimized feature selection techniques. Various classification models, including Random Forest, XGBoost, CatBoost, SVM, and Transformer, are employed to predict defects. To further enhance model performance, metaheuristic-based feature selection methods such as the Bat Algorithm (BA) and Particle Swarm Optimization (PSO) are applied to reduce dimensionality and improve predictive relevance. Experimental results show that Random Forest with BA optimization achieves the highest performance, with an F1-score of 0.83 and an AUC-ROC of 0.86, outperforming other models. While the Transformer model does not surpass tree-based algorithms in all metrics, it shows high recall and competitive F1-scores, making it suitable for high-sensitivity applications. These findings highlight the importance of integrating optimized machine learning models and feature selection techniques to improve model robustness, reduce computational complexity, and meet the needs of Agile development. This approach supports software teams in prioritizing quality assurance tasks, reducing long-term maintenance costs, and optimizing defect management processes.

Keywords: agile software practices; bug prediction; defect classification; feature selection; metaheuristic optimization

How to Cite: Faiq Dhimas Wicaksono and Daniel Siahaan, "Enhancing Agile Defect Prediction with Optimized Machine Learning and Feature Selection", *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 9, no. 4, pp. 814 - 819, Aug. 2025.
Permalink/DOI: <https://doi.org/10.29207/resti.v9i4.6713>

Received: June 2, 2025

Accepted: July 23, 2025

Available Online: August 18, 2025

*This is an open-access article under the CC BY 4.0 License
Published by Ikatan Ahli Informatika Indonesia*

1. Introduction

Software defects pose a major concern in the software industry since they can lead to functional failures, security vulnerabilities, and financial losses. These defects arise from various factors, including poor design, coding errors, and inadequate testing processes. It has been observed in various industry reports that bugs that slip through during the early development phase may cause huge financial impacts, project delays, and end-user dissatisfaction [1]. From a project management oversight perspective, it is important to manage such issues since open bugs can hold up progress, waste resources, and compromise the reliability of the final product [2].

With increasing and iterative delivery cycles, Agile development practices have become the norm in modern software engineering [3]. Although Agile's

iterative nature adds complexity to maintaining quality and readability of growing codebases [4], the shortened development cycles and frequent releases make early bug detection and prediction essential for preserving software integrity and reducing remediation costs [5]. Early bug detection allows development teams to take corrective action timely, minimize system failures, and enhance user trust in the software product [6].

Several experiments have explored machine learning techniques to predict software defects [7], but these approaches commonly depend on static data such as code metrics or commit history [8], [9]. These approaches often fail to capture the richer, contextual information in real-world artifacts such as issue reports, system logs, and bug documentation. Though numerous studies have tested classification and feature selection methods [10], few have clearly aligned their

methodologies to the dynamic nature of Agile development or implemented high-level optimization techniques to improve prediction accuracy [11], [12].

To address the above-mentioned gaps, this study proposes a defect prediction model for Agile software development, utilizing machine learning and deep learning techniques. Algorithms such as Random Forest (RF), Support Vector Machine (SVM), XGBoost, CatBoost, and Transformer are selected because of their strength in processing high-dimensional and imbalanced datasets [13]. To optimize the model further, metaheuristic feature selection methods such as the Bat Algorithm (BA) and Particle Swarm Optimization (PSO) are utilized to identify the most relevant input variables, thereby improving accuracy and computational efficiency [14].

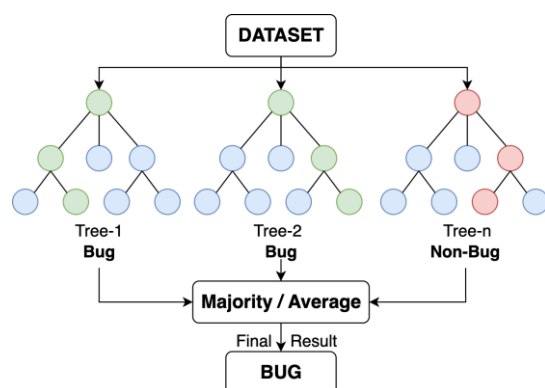


Figure 1. Random Forest Architecture

Random Forest is a strong defect prediction classifier with its architecture illustrated in Figure 1, particularly for balanced datasets [15], and this supports our choice

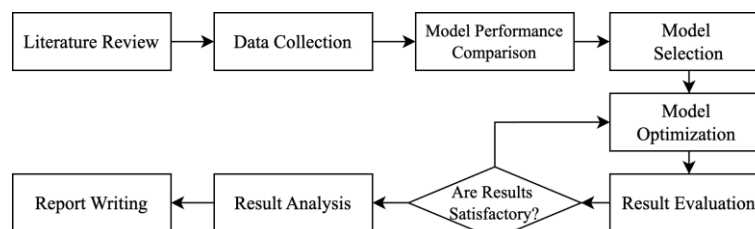


Figure 2. Research Process

2.1 Data Collection

The Jira dataset used in this study was obtained from the Tokopedia project in 2022 and its access is restricted due to confidentiality agreements. For reproducibility, similar datasets can be acquired from Jira by querying specific data via REST API or by manually exporting data using the "Issues" search filters in Jira.

The dataset consists of 980 issue tickets collected between March 7 and March 20, 2022. Each record contained attributes such as Project Name, Project Type, Priority, Assignee, Reporter, Creator, Created, Status, Project Key, Project Lead, Updated, Resolved, Components, Labels, Custom Fields, Parent, Parent Summary, Description, TE Assignee, Step to Reproduce, Watchers, Comments, and Issue Type.

to leverage it for further optimization after initial comparative evaluation. We also included SVM, boosting algorithms, and Transformer-based models to broaden the analysis, as these methods have proven effective in software tasks [16]. SVM is effective in high-dimensional classification tasks [9], while XGBoost and CatBoost offer strong performance with complex and categorical data, with minimal preprocessing [17]. Transformer models, including those used in vulnerability detection, are well-suited to understand unstructured data like bug reports with deep contextual learning. Like hybrid models with attention can capture semantic and contextual relationships well in code-mixed text [18], Transformers are suitable for leveraging attention to identify subtle patterns in noisy software bug data [19]. Including these models ensures a comprehensive comparison across algorithmic strengths and data types.

This research's prime contribution is the creation of a context-aware, high-performance bug prediction framework that integrates real project data and advanced optimization techniques. This model is expected to help development teams reduce maintenance costs, improve software quality, and accelerate delivery without compromising reliability by supporting early defect detection within Agile workflows.

2. Research Methods

Figure 2 provides an overview of the methodology used to build and validate the proposed bug prediction model, including data acquisition, preprocessing, model selection, feature optimization, and evaluation.

These fields provided contextual insights into how issues were handled, including hierarchical relations, responsible individuals, and resolution timelines [20].

Each attribute was described in terms of data type, example values, and relevance to bug prediction. This descriptive analysis facilitated the understanding, transformation, and selection of features. Components such as Priority, Status, and Assignee showed notable variation and were therefore considered potentially influential in bug occurrence.

Exploratory Data Analysis (EDA) was implemented to analyze the distribution and correlations within the dataset [21]. Key features, including priority, status, and time-to-resolution, were visualized using histogram and heatmap techniques [22].

Correlation heatmaps revealed that features like Assignee, Reporter, and Components had significant associations with the bug label, as shown in Figure 3. Priority was strongly positively correlated with bug presence, while certain assignees and reporters demonstrated inverse associations. In the heatmaps, darker colors indicate stronger positive or negative correlations, whereas lighter shades represent weaker or negligible relationships. This analysis guided initial assumptions on feature relevance.

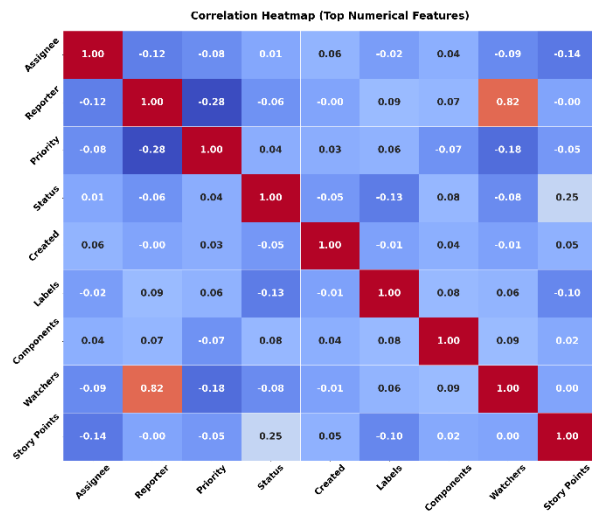


Figure 3. Heatmaps Correlation Features

2.2. Data Preprocessing

Initial preprocessing involved handling missing values by filling categorical columns using mode or "unknown," and imputing numerical values with the median. Text attributes such as Description, Step to Reproduce, and Comments were converted to binary indicators. Categorical fields were encoded using Label Encoding, converting textual values into numerical form. For instance, 'High', 'Medium', and 'Low' priorities were mapped to 2, 1, and 0, respectively [23].

Date fields such as Created, Updated, and Resolved were transformed into UNIX timestamps to ensure compatibility with machine learning algorithms [24]. Duplicate entries were removed, and normalization was applied using StandardScaler [25]. Additionally, correlation tests (Chi-square, Pearson, and Spearman) were performed to identify attributes with statistically significant correlations about bug labels [26].

2.3 Model Development and Selection

Multiple classification models, including RF, SVM, XGBoost, CatBoost, and Transformer, were developed using the preprocessed dataset. These models were selected for their effectiveness in handling complex and large-scale datasets [12], [19]. The process is summarized in Figure 4.

The models were evaluated using accuracy, recall, AUC-ROC, precision, and F1-score, prioritizing accuracy and recall to reduce misclassified bugs [27].

Model consistency was assessed through cross-validation, with hyperparameter tuning applied when initial results were unsatisfactory. In cases where performance differences were minimal (under 0.05), the selection was based on the balance of metrics, validation stability, computational efficiency, and ease of deployment [28]. The final model demonstrated strong predictive performance and was chosen for further optimization in dynamic software development.

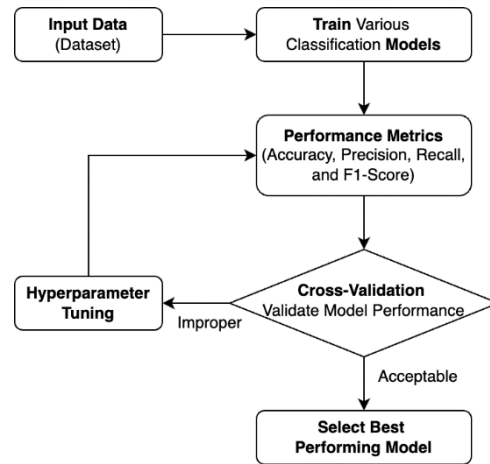


Figure 4. Classification Model Selection Flow

2.4 Feature Selection and Model Optimization

After identifying the best-performing model, feature selection was applied using the Bat Algorithm and Particle Swarm Optimization. These metaheuristic techniques, which are famously based on biological systems, are highly effective at selecting the optimal feature subsets, improving predictive accuracy, and reducing the model complexity [12], [27]. Figure 5 provides a detailed view of the BA design architecture.

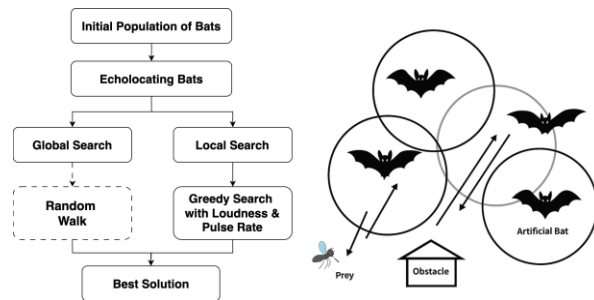


Figure 5. Bat Algorithm Design for Optimization

In order to determine the more suitable approach, both algorithms were executed using the same parameter values and under the same experimental conditions [29]. For the Bat Algorithm, the settings included 15 bats, 20 generations, an initial alpha value of 0.9 (decaying gradually over iterations), and a gamma value of 0.9. The Particle Swarm Optimization was configured with 15 particles, 20 iterations, an inertia weight (w) of 0.72, and cognitive and social coefficients ($c1$ and $c2$) each set to 1.49. These parameter values were selected based on commonly recommended defaults in the literature and to maintain a balance

between search performance and computational efficiency. Each algorithm generated a feature subset, which was then used to train a baseline classification model and compared on accuracy, recall, and computational efficiency for a meaningful evaluation. The method that resulted in improved performance was selected for optimization and subsequent analysis [30].

2.5 Model Training

The optimized data was subjected to an 80:20 train-test split to prepare it for supervised learning. Hyperparameter tuning and k-fold cross-validation were utilized in the training process to improve generalization and reduce the overfitting threat. The model was trained using the feature subset obtained from the selection process.

To assess its performance, evaluation metrics such as accuracy, AUC-ROC, confusion matrix, and a detailed classification report were utilized to examine the model's performance in accurately detecting and classifying software defects [8].

2.6 Evaluation Metrics and Validation

Model performance was confirmed using k-fold cross-validation to minimize overfitting and maximize model generalizability. Performance was evaluated using conventional classification metrics like accuracy, recall, AUC, precision, F1-score, and confusion matrix analysis.

While no absolute cutoffs were used, a general performance benchmark of around 80% (or 0.80) on these metrics was employed as a reference point to determine that the model met a minimum reliability threshold, particularly in minimizing false negatives and guaranteeing classification consistency.

2.7 Result Analysis

This stage assessed the model's performance by analyzing classification metrics and examining the confusion matrix results. Misclassification trends were reviewed to identify areas where the model underperformed.

Error analysis also provided further insight into bias or imbalances, informing targeted improvements such as hyperparameter tuning and more precise feature selection.

3. Results and Discussions

We collected 980 Jira issue tickets between March 7 and March 20, 2022. Each ticket included metadata such as when it was created, who it was assigned to, an estimated complexity level, and a short description. Preprocessing included handling missing data, encoding categories, simplifying text fields, and removing features that could cause data leakage. The data was normalized and divided into train and test sets.

3.1 Baseline Model Performance

Five classifiers (RF, SVM, XGBoost, CatBoost, and Transformer) were performed to evaluate model performance. The evaluation applied 5-fold cross-validation to minimize performance variance. Key indicators included accuracy, recall, AUC-ROC, precision, and F1-score.

According to Table 1, Random Forest demonstrated the most balanced and consistent performance across all metrics. It achieved an accuracy and precision of 0.78, a recall of 0.87, and the highest F1-score (0.82) and AUC-ROC (0.84). The SVM model, while recording the highest recall (0.98), showed noticeably lower accuracy (0.62) and precision (0.61). This suggests that although SVM was very sensitive in detecting potential defects, it also produced a high rate of false positives and an overall less balanced performance profile.

Table 1. Model Performance Comparison

Model	Accuracy	Precisior	Recall	F1-Score	AUC-ROC
RF	0.78	0.78	0.87	0.82	0.84
SVM	0.62	0.61	0.98	0.75	0.67
XGBoost	0.75	0.75	0.84	0.80	0.79
CatBoost	0.74	0.75	0.84	0.79	0.82
Transformer	0.70	0.71	0.84	0.77	0.78

XGBoost and CatBoost both performed well, with recall values of 0.84 and F1-scores close to 0.80. CatBoost achieved a slightly higher AUC-ROC (0.82) compared to XGBoost (0.79), indicating strong discriminatory ability. These findings illustrate that boosting models were effective in capturing defect patterns but did not surpass RF in overall balance. The Transformer model achieved recall comparable to the boosting models (0.84) and an F1-score of 0.77. Although it demonstrated good sensitivity and the ability to model complex patterns, its accuracy (0.70) and precision (0.71) were relatively lower than the tree-based classifiers, reflecting less consistent predictions.

3.2 Classification Model Selection

In determining the most suitable model for further optimization, several factors were considered beyond raw performance metrics. Although SVM achieved the highest recall, its low precision and AUC-ROC indicated a tendency to generate many false positives, which could limit its practical use.

XGBoost and CatBoost delivered competitive results, with recall and AUC values close to RF. However, their training times and complexity were less favorable for iterative development. The Transformer also showed good sensitivity but had lower precision and required more computational resources to train effectively.

Considering the balance between all metrics, Random Forest was selected as the baseline model. Its consistent performance, simplicity, and robustness made it the most appropriate choice for this study.

3.3 Feature Optimization

As shown in Tables 2 and 3, Bat Algorithm (BA) and Particle Swarm Optimization (PSO) were utilized in feature selection for improving the performance of the Random Forest model.

Table 2. RF Result after BA Feature Selection

Type	Precision	Recall	F1-Score	Support
Non-Bug	0.76	0.72	0.74	81
Bug	0.82	0.84	0.83	116
Accuracy			0.79	197
AUC-ROC				0.86

Table 3. RF Result after PSO Feature Selection

Type	Precision	Recall	F1-Score	Support
Non-Bug	0.74	0.72	0.73	81
Bug	0.81	0.83	0.82	116
Accuracy			0.80	197
AUC-ROC				0.85

Both BA and PSO improved the model's classification accuracy. BA achieved a higher F1-score (0.83) and AUC (0.86), while PSO also showed significant improvements with slightly lower metrics.

3.4 Post-Optimization Evaluation

The results confirmed the effectiveness of the optimization process compared to the baseline Random Forest model before feature selection, as shown in Table 4. Although overall accuracy remained consistent, precision, F1-score, and AUC-ROC showed clear improvements. These outcomes indicate that applying feature selection and parameter tuning helped reduce irrelevant data and emphasize influential variables, making the model more capable of distinguishing between bug and non-bug cases.

Table 4. Evaluation Results After Model Optimization

Condition	Accuracy	Precision	Recall	F1-Score	AUC-ROC
RF baseline	0.78	0.78	0.87	0.82	0.84
RF + Bat	0.79	0.82	0.84	0.83	0.86
RF + PSO	0.80	0.81	0.83	0.82	0.85

The rise in precision indicates fewer false positives, and the increases in F1-score and AUC-ROC signal better balance and classification strength. These gains make the optimized model well-suited for Agile development, supporting early bug detection, better code quality, faster delivery, and streamlined workflows.

This study has several limitations to consider when interpreting the results. First, the dataset is proprietary and confidential, as it was derived from Jira issue tickets within the Tokopedia. This restricts direct access for reproducibility and external validation. Future studies could address this by using comparable public datasets or generating synthetic data to create similar conditions. Second, the analysis focuses on Agile-based practices, which means the findings may not apply to other SDLC approaches. Third, this work is limited to predicting defects at the ticket level rather than real-time or code-level prediction. Acknowledging these limitations can help guide and strengthen future studies in this area.

4. Conclusions

This study shows the effectiveness of machine learning and deep learning approaches, particularly when combined with metaheuristic-based feature selection, in predicting software defects using issue ticket data from Jira. Five classification models (RF, SVM, XGBoost, CatBoost, and Transformer) were evaluated, with the Transformer representing the latest deep learning architecture. The findings reveal that Random Forest achieved the most stable and balanced performance before optimization, with an F1-score of 0.82 and an AUC-ROC of 0.84. On the other hand, the Transformer model achieved strong recall of 0.84 and an AUC-ROC of 0.78, although its precision and accuracy were lower than Random Forest. This is still considered promising because in settings where it is more important to identify as many defect cases as possible, recall often takes priority over overall discrimination performance. The Transformer also has the advantage of capturing complex relationships in the data, which is useful for Jira issue tickets, though it generally requires more data and training time.

Feature selection using the Bat Algorithm and Particle Swarm Optimization led to improved model performance. A Random Forest model optimized using BA achieved the highest results, with an F1-score of 0.83 and an AUC-ROC of 0.86. PSO-based optimization produced comparable results, with an F1-score of 0.82 and an AUC-ROC of 0.85. The findings confirm the applicability of metaheuristic approaches in identifying important features without compromising model simplicity and generalization improvement.

Feature contribution analysis revealed that project type, priority, assigned personnel, and component category were highly significant for defect prediction. The results underscore the importance of utilizing systematic project management data within Agile processes. The model suggested supports early detection of bugs, which is vital to ensure software quality, shorten release cycles, and minimize long-term maintenance efforts. This work offers an efficient and scalable defect prediction solution, further propelling intelligent, context-aware tools in modern software engineering.

Acknowledgments

The authors appreciate Institut Teknologi Sepuluh Nopember's (ITS) support and the use of institutional resources in this research.

References

- [1] S. Haldar and L. F. Capretz, "Explainable Software Defect Prediction from Cross Company Project Metrics using Machine Learning," in *Proceedings of the 7th International Conference on Intelligent Computing and Control Systems, ICICCS 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 150–157. doi: 10.1109/ICICCS56967.2023.10142534.
- [2] U. S. B and R. Sadam, "How Far Does the Predictive Decision Impact the Software Project? The Cost, Service Time, and

- Failure Analysis from a Cross-Project Defect Prediction Model,” *Journal of Systems and Software*, vol. 195, no. January, p. 111522, Sep. 2023, doi: 10.1016/j.jss.2022.111522.
- [3] R. Fauzan, M. Z. Hamidi, W. A. Safitri, D. O. Siahaan, and M. I. Karimi, “Ontology in Requirements Software Development Method: A Systematic Literature Review,” *Journal of Information Technology and Cyber Security*, vol. 3, no. 1, pp. 14–32, Jan. 2025, doi: 10.30996/jitcs.12297.
- [4] M. A. Al Hilmi, A. Puspaningrum, Darsih, D. O. Siahaan, H. S. Samosir, and A. S. Rahma, “Research Trends, Detection Methods, Practices, and Challenges in Code Smell: SLR,” *IEEE Access*, vol. 11, pp. 129536–129551, 2023, doi: 10.1109/ACCESS.2023.3334258.
- [5] A. Fatwanto, M. Nur Aslam, R. Ndugi, and M. Syafrudin, “An Investigation Towards Resampling Techniques and Classification Algorithms on CM1 NASA PROMISE Dataset for Software Defect Prediction,” *Jurnal RESTI*, vol. 8, no. 5, pp. 631–643, Oct. 2024, doi: 10.29207/resti.v8i5.5910.
- [6] Ö. F. Arar and K. Ayan, “Software defect prediction using cost-sensitive neural network,” *Applied Soft Computing Journal*, vol. 33, pp. 263–277, Apr. 2015, doi: 10.1016/j.asoc.2015.04.045.
- [7] C. W. Yohannese, T. Li, and K. Bashir, “A Three-Stage Based Ensemble Learning for Improved Software Fault Prediction: An Empirical Comparative Study,” *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, pp. 1229–1247, Jul. 2018, doi: 10.2991/ijcis.11.1.92.
- [8] M. Z. F. N. Siswanto and U. L. Yuhana, “Software Defect Prediction Based on Optimized Machine Learning Models: A Comparative Study,” *Teknika*, vol. 12, no. 2, pp. 166–172, Jun. 2023, doi: 10.34148/teknika.v12i2.634.
- [9] M. Cetiner and O. K. Sahingoz, “A Comparative Analysis for Machine Learning based Software Defect Prediction Systems,” in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2020, pp. 1–7, doi: 10.1109/ICCCNT49239.2020.9225352.
- [10] N. AlNuaimi, M. M. Masud, M. A. Serhani, and N. Zaki, “Streaming feature selection algorithms for big data: A survey,” *Applied Computing and Informatics*, vol. 18, no. 1–2, pp. 113–135, Jan. 2022, doi: 10.1016/j.aci.2019.01.001.
- [11] N. A. A. Khleel and K. Nehez, “Comprehensive Study on Machine Learning Techniques for Software Bug Prediction,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 8, pp. 726–735, 2021, doi: 10.14569/IJACSA.2021.0120884.
- [12] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, “Software defect prediction using feature selection and random forest algorithm,” in *Proceedings - 2017 International Conference on New Trends in Computing Sciences, ICTCS 2017*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 252–257, doi: 10.1109/ICTCS.2017.39.
- [13] B. R. P. Darnoto, D. Siahaan, and D. Purwitasari, “Automated Detection of Persuasive Content in Electronic News,” *Informatics*, vol. 10, no. 4, Dec. 2023, doi: 10.3390/informatics10040086.
- [14] D. Ariatmanto and A. M. Rifai, “The Impact of Feature Extraction in Random Forest Classifier for Fake News Detection,” *Jurnal RESTI*, vol. 8, no. 6, pp. 730–736, Dec. 2024, doi: 10.29207/resti.v8i6.6017.
- [15] D. I. Ulumi and D. Siahaan, “Weighted knn using grey relational analysis for cross-project defect prediction,” in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Sep. 2019, doi: 10.1088/1742-6596/1230/1/012062.
- [16] D. Siahaan et al., “A scoping review of auto-generating transformation between software development artifacts,” 2023, *Frontiers Media SA*, doi: 10.3389/fcomp.2023.1306064.
- [17] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, “A comparative analysis of gradient boosting algorithms,” *Artif Intell Rev*, vol. 54, no. 3, pp. 1937–1967, Mar. 2021, doi: 10.1007/s10462-020-09896-5.
- [18] M. A. Rosid, D. Siahaan, and A. Saikhu, “Sarcasm Detection in Indonesian-English Code-Mixed Text Using Multihead Attention-Based Convolutional and Bi-Directional GRU,” *IEEE Access*, 2024, doi: 10.1109/ACCESS.2024.3436107.
- [19] C. Thapa, S. I. Jang, M. E. Ahmed, S. Camtepe, J. Pieprzyk, and S. Nepal, “Transformer-Based Language Models for Software Vulnerability Detection,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery (ACM), Apr. 2022, pp. 481–496, doi: 10.1145/3564625.3567985.
- [20] Mikko Raatikainen et al., “Improved Management of Issue Dependencies in Issue Trackers of Large Collaborative Projects,” *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2128–2148, Apr. 2023, doi: 10.1109/TSE.2022.3212166.
- [21] K. Sahoo, A. K. Samal, J. Pramanik, and S. K. Pani, “Exploratory data analysis using python,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 12, pp. 4727–4735, Oct. 2019, doi: 10.35940/ijitee.L3591.1081219.
- [22] M. Ichino, “Exploratory Analysis of Distributional Data Using the Quantile Method,” *AppliedMath*, vol. 4, no. 1, pp. 261–288, Mar. 2024, doi: 10.3390/appliedmath4010014.
- [23] F. Bolikulov, R. Nasimov, A. Rashidov, F. Akhmedov, and Y. I. Cho, “Effective Methods of Categorical Data Encoding for Artificial Intelligence Algorithms,” *Mathematics*, vol. 12, no. 16, Aug. 2024, doi: 10.3390/math12162553.
- [24] R. Ambe, “Classification and quantification of timestamp data quality issues and its impact on data quality outcome,” *Data Intell*, vol. 6, no. 3, pp. 812–833, Dec. 2023, doi: 10.1162/dint_a.00238.
- [25] K. Mahmud Sujon, R. Binti Hassan, Z. Tusnia Towshi, M. A. Othman, M. Abdus Samad, and K. Choi, “When to Use Standardization and Normalization: Empirical Evidence From Machine Learning Models and XAI,” *IEEE Access*, vol. 12, pp. 135300–135314, 2024, doi: 10.1109/ACCESS.2024.3462434.
- [26] M. Y. A. Pratama, R. Herteno, M. R. Faisal, R. A. Nugroho, and F. Abadi, “Improving with Hybrid Feature Selection in Software Defect Prediction,” *Jurnal Online Informatika*, vol. 9, no. 1, pp. 52–60, Apr. 2024, doi: 10.15575/join.v9i1.1307.
- [27] A. Arya and S. K. Malik, “An Improved Firefly-Based Feature Selection Method for Software Fault Identification and Classification,” *International Journal of Performability Engineering*, vol. 19, no. 11, pp. 744–752, Nov. 2023, doi: 10.23940/ijpe.23.11.p5.744752.
- [28] B. Bischl et al., “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges,” *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 13, no. 2, Mar. 2023, doi: 10.1002/widm.1484.
- [29] M. Martinović, K. Dokic, and D. Pudić, “Comparative Analysis of Machine Learning Models for Predicting Innovation Outcomes: An Applied AI Approach,” *Applied Sciences (Switzerland)*, vol. 15, no. 7, Apr. 2025, doi: 10.3390/app15073636.
- [30] A. I. Adler and A. Painsky, “Feature Importance in Gradient Boosting Trees with Cross-Validation Feature Selection,” *Entropy*, vol. 24, no. 5, May 2022, doi: 10.3390/e24050687.