Just-In-Time Bug Prediction Framework Using Data Analytics, Soft Computing, and Deep Kendall Analysis

Veena Janardhan Jadhav[1], Dr. Prakash Devale[2], Dr. Rohini Jadhav[3], Dr. Suhas Patil[4]
[1]*Assistant Professor, Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune*
[2,4]*Professor, Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune*
[3]*Associate Professor, Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune*

Abstract— **Just-In-Time (JIT) bug prediction improves software quality by identifying defect-prone commits at submission time, yet existing methods suffer from class imbalance, weak representation of code change semantics, high false-positive rates, and limited explainability. Motivated by these challenges, this work proposes an interpretable JIT bug prediction framework integrating data balancing, structural feature learning, and deep sequence modeling. A Density-aware Borderline Synthetic Tomek (DBST) cleanup mechanism addresses class imbalance by preserving critical boundary instances while improving class separation. An Explainable Backfit Structural Transformer captures syntactic and semantic code change structures using dependency and constituency parsing with equivariant semantic attention. Additionally, an Explainable Deep Kendall Analysis module employs a Deep Loopy Bi-LSTM to model long-term commit dependencies and rank-based correlations, reducing false positives. A global-local, model-agnostic interpretability mechanism further provides instance-level explanations. The proposed framework enhances prediction reliability, interpretability, and practical applicability in real-world software development workflows.**

Index Terms— **class imbalance, explainable deep learning, Just-in-Time bug prediction, Kendall correlation, structural code analysis**

I. INTRODUCTION

Early identification of software defects is essential for reducing development cost and maintaining high softwarequality[1],[14]. Just-In-Time (JIT) bug prediction addresses this need by evaluating the risk of code changes at the moment they are committed, enabling development teams to prioritise code reviews and testing activities more effectively [2], [11]. By analysing information derived from code modifications and commit histories, JIT models provide timely insights that support efficient and proactive quality assurance [2],[16].

Despite considerable progress, existing JIT software defect prediction approaches still face important limitations. JIT datasets are typically highly imbalanced, with defective commits representing only a small fraction of the data, which often biases learning algorithms and degrades generalization performance [4],[17]. In addition, many approaches rely on shallow or coarse-grained features that fail to capture the underlying syntactic and semantic structure of code changes [8],[11]. The increasing use of complex learning models has further introduced challenges related to interpretability, as many predictions are difficult for practitioners to understand or trust in practical development settings [10],[18].

To address these challenges, this paper proposes an integrated JIT bug prediction framework that jointly emphasizes robustness, accuracy, and interpretability. The framework combines density-aware data cleanup to handle class imbalance, structured feature extraction to model code semantics and syntax, and explainable dependency analysis to reduce false positives while providing transparent prediction rationale. Together, these components aim to improve the practical reliability and adoption of JIT bug prediction in real-world software engineering workflows.

II. RELATED WORK

Software defect prediction has been extensively studied to improve software quality and reduce maintenance costs. Early research primarily focused on module-level defect prediction, using static software metrics such as code size, complexity, and historical defect data with classical machine learning classifiers [1],[4] including Naïve Bayes, Decision Trees, Support Vector Machines, and Random Forests. Although these approaches demonstrated reasonable accuracy, they suffered from poor generalization, project dependency, and severe class imbalance issues [3],[7].

To address these challenges, researchers introduced soft computing and hybrid models, such as fuzzy logic systems, neuro-fuzzy frameworks, ensemble learning, and evolutionary optimization techniques. These approaches improved prediction accuracy and uncertainty handling but increased model complexity and relied heavily on static metrics, limiting their ability to capture semantic characteristics of code changes.

Class imbalance has been identified as a critical limitation in defect prediction. Oversampling techniques such as SMOTE and its variants were widely adopted to rebalance datasets [5]. While these methods improved recall, they frequently generated noisy synthetic instances near class boundaries, resulting in elevated false-positive rates [6],[17]. Conversely, undersampling methods often removed informative majority-class instances, degrading overall performance.

More recent studies have focused on Just-In-Time (JIT) bug prediction, which aims to identify defect-prone commits at commit time using change-level features [2],[11] such as code churn, developer activity, and historical changes. Large-scale datasets such as ApacheJIT enabled the application of advanced machine learning and deep learning models for JIT prediction. However, JIT datasets remain highly imbalanced, noisy, and sensitive to boundary instances, making effective data preprocessing essential.

To enhance feature representation, several works employed deep learning models, including autoencoders, convolutional neural networks, graph-based neural networks, and Bi-LSTMs, to capture syntactic, semantic, and temporal characteristics of code changes [8],[9]. Although these models achieved performance improvements, they often produced high false-positive rates, required extensive labeled data, and lacked transparency.

Explainability has therefore emerged as a key requirement for practical adoption. Model-agnostic explanation techniques such as LIME and SHAP have been applied to defect prediction [12],[13]; however, these methods frequently fail to generate explanations that align with developer intuition, particularly in JIT settings [10],[18] where synthetic neighbors may not accurately represent real commit behavior.

Overall, existing literature highlights persistent limitations in handling class imbalance without boundary distortion, capturing deep syntactic and semantic relationships, modeling long-term commit dependencies, and providing trustworthy, instance-level explanations. These gaps motivate the need for integrated and interpretable JIT bug prediction frameworks.

III. PROPOSED METHODOLOGY

Modern Just-In-Time (JIT) bug prediction systems often struggle with three practical issues: highly imbalanced datasets, weak representation of code change semantics, and poor explainability. To overcome these challenges, this work proposes an end-to-end, explainable JIT bug prediction framework that integrates data cleanup, structure-aware feature learning, deep sequential modeling, and transparent explanations into a unified pipeline.

First, the raw JIT commit data is processed using a Density-aware Borderline Synthetic Tomek (DBST) cleanup mechanism [5],[6],[17]. Instead of treating boundary samples as noise, DBST preserves and refines them using density-based clustering, Borderline SMOTE, and Tomek link removal. This step effectively balances the dataset while maintaining meaningful decision boundaries, addressing a major limitation of traditional oversampling techniques.

An Explainable Backfit Structural Transformer extracts both syntactic and semantic features from code changes. Dependency and constituency parsing capture how code elements are structurally related, while an equivariant semantic attention mechanism models the intent and meaning behind code modifications [8],[11]. The backfitting process further refines important features, ensuring that only relevant structural patterns influence prediction.

The extracted features are then passed to a Deep Kendall Analysis module, built using a Deep Loopy Bidirectional LSTM [9],[16]. This component captures long-range and temporal dependencies across commit histories and ranks commits based on defect risk. Kendall's rank correlation is used to stabilize predictions and reduce false positives, making the model more suitable for real-world code review prioritization.

Finally, a global–local model-agnostic explanation layer generates human-interpretable explanations [12],[13],[18]. It highlights why a specific commit is flagged as risky (local view) and reveals overall defect patterns across the project (global view). This ensures that the model does not behave as a black box and supports developer trust and adoption.

Overall, the proposed methodology delivers a balanced, structure-aware, and explainable JIT bug prediction system, directly addressing the shortcomings identified in existing bug prediction research.

IV. MATHEMATICAL MODEL

This section presents the mathematical formulation of the proposed Just-In-Time (JIT) bug prediction framework, comprising density-aware data cleanup, structural feature learning, and ranking-oriented defect prediction.

A. Problem Definition

Let the JIT dataset be defined as:

$$D = \{(x_i, y_i)\}_{i=1}^{N}, y_i \in \{0,1\}$$

where x_i represents a commit instance and y_i denotes its defect label (buggy or clean). Given the severe class imbalance ($y = 1 \ll y = 0$), the objective is to learn a function:

$$f: x_i \to P(y_i = 1 \mid x_i)$$

that accurately predicts and ranks defect-inducing commits.

B. Density-Aware DBST Cleanup Mechanism

To address imbalance and boundary noise, a Density-aware Borderline Synthetic Tomek (DBST) mechanism is applied.

The taxicab (Manhattan) distance between two commit vectors is defined as:

$$d(x_i, x_j) = \sum_k \mid x_{ik} - x_{jk} \mid$$

The local relative density of a commit x_i is computed as:

$$\rho_i = \sum_{x_j \in N_i} \frac{1}{d(x_i, x_j)}$$

Low-density instances near class boundaries are treated as critical learning points. Borderline SMOTE generates synthetic minority samples near these regions, while Tomek Links identify and remove overlapping majority–minority pairs.

The resulting balanced dataset is denoted as:

$$D' = \mathrm{DBST}(D)$$

C. Structural Feature Representation

Each commit is represented through joint syntactic and semantic modeling using an Explainable Backfit Structural Transformer.

The syntactic relationship between code elements c_i and c_j is expressed as:

$$S_{ij} = f(c_i, c_j)$$

capturing dependency and constituency structures.

Semantic representation is obtained using an equivariant attention mechanism:

$$E_j = \sum_i \alpha_{ji} \cdot h_i$$

where α_{ji} denotes attention weights and h_i represents latent code embeddings.

The final commit representation is formed as:

$$Z = [S \parallel E]$$

Backfitting iteratively refines feature contributions, enhancing stability and interpretability.

D. Deep Kendall-Based Bug Prediction

To model temporal dependencies across code changes, the extracted representations are passed to a Deep Loopy Bi-LSTM:

$$h_t = \mathrm{BiLSTM}(Z_t, h_{t-1})$$

Bidirectional processing captures both historical and forward contextual information.

The probability of a commit being defective is estimated as:

$$P(y = 1 \mid Z) = \sigma(Wh_t + b)$$

where $\sigma(\cdot)$ is the sigmoid activation.

E. Ranking Evaluation Using Kendall's Tau

Rather than binary classification alone, commits are ranked by predicted defect risk [16]. The agreement between predicted and actual rankings is measured using Kendall's Tau [15]:

$$\tau = \frac{C - D}{C + D}$$

where C and D denote concordant and discordant commit pairs, respectively. A higher τ indicates more reliable prioritization of high-risk commits for code review.

V.CONCLUSION

This work presents an interpretable JIT bug prediction framework that effectively addresses class imbalance, captures structural and semantic code change information, and reduces false positives. The DBST cleanup mechanism improves data quality, while deep sequential modeling with Kendall-based ranking supports reliable commit prioritization [6],[17]. By providing clear, human-interpretable explanations, the proposed approach enhances practical usability and developer trust, making it well suited for real-world software development workflows [16],[18]. Top ofForm Bottom of Form

REFERENCES

[1] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2–13, 2007.

[2] S. Kim, E. J. Whitehead Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" IEEE Transactions on Software Engineering, vol. 34, no. 2, pp. 181–196, 2008.

[3] J. Nam and S. Kim, "Heterogeneous defect prediction," Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 508–519, 2015.

[4] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, pp. 137–144, 2010.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.

[6] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," ACM SIGKDD Explorations, vol. 6, no. 1, pp. 20–29, 2004.

[7] Z. Li, X. Jing, X. Zhu, H. Zhang, and B. Xu, "Heterogeneous defect prediction with two-stage ensemble learning," Automated Software Engineering, vol. 25, pp. 1–46, 2018.

[8] J. Hoang, S. Kamei, D. Lo, and E. Shihab, "DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction," Proceedings of the 2019 IEEE/ACM International Conference on Mining Software Repositories (MSR), pp. 34–45, 2019.

[9] Z. Chen, W. Zhang, J. Chen, and X. Yang, "Just-in-time defect prediction via deep learning," Information and Software Technology, vol. 109, pp. 1–15, 2019.

[10] Y. Wang, G. Li, T. Liu, and H. Chen, "Explainable defect prediction with attention-based neural networks," IEEE Access, vol. 9, pp. 128923–128935, 2021.

[11] S. Panichella, A. Zaidman, and A. Bacchelli, "The impact of code changes on software quality: A large-scale empirical study," Empirical Software Engineering, vol. 26, no. 4, 2021.

[12] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," Advances in Neural Information Processing Systems (NeurIPS), pp. 4765–4774, 2017.

[13] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144, 2016.

[14] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed., Morgan Kaufmann, 2012.

[15] M. Kendall, "A new measure of rank correlation," Biometrika, vol. 30, no. 1–2, pp. 81–93, 1938.

[16] Y. Li, Y. Wang, and M. Harman, "Improving just-in-time defect prediction by ranking change risk," IEEE Transactions on Software Engineering, vol. 49, no. 2, pp. 1043–1058, 2023.

[17] H. Tong, X. Xia, D. Lo, and S. Li, "Revisiting class imbalance in software defect prediction," ACM Transactions on Software Engineering and Methodology, vol. 31, no. 3, 2022.

[18] K. Zhang, S. Wang, and D. Lo, "Interpretable deep learning for software defect prediction,"

IEEE Transactions on Reliability, vol. 72, no. 1, pp. 45–60, 2023.

[19] V. J. Jadhav, P. Devale, R. Jadhav, M. Molawade, S. Mohite, and R. V. Bidwe, “Bug predictive models based on data analytics and soft computing techniques: A survey,” in Proceedings of the 10th International Conference on Computing for Sustainable Global Development (INDIACom), Mar. 2023, pp. 1–11.

[20] V. J. Jadhav, P. Devale, and A. Kadam, “JIT software bug prediction using DBST cleanup mechanism with explainable backfit transformer and deep Kendal lanalysis,”in Proceedings of the 12th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2025, pp. 1936–1943.