Predicting Software Bug Resolution Time: A Comparative Study of Machine Learning Algorithms

Harun Kunovac
Department of Information Technologies
International Burch University
Bosnia and Herzegovina

Zerina Altoka
Department of Information Technologies
International Burch University
Bosnia and Herzegovina

ABSTRACT
Software maintenance is one of the costliest activities in the software development process, and bug fixing is among the most time-consuming. Time estimation for bug fixes is a major issue for developers and project managers, as it directly affects task order, release planning, and customer satisfaction. This study investigates the prediction of bug resolution time by classifying bugs into fast and slow groups using machine learning approaches. Publicly available issue tracking datasets are utilized, with structured metadata features (e.g., severity, priority, and comments count) and textual features from bug report summaries. Textual features were preprocessed via NLP methods like TF-IDF vectorization and text embeddings, depending on the model type. RandomForest, LogisticRegression, LightGBM, SGD Classifier, and Multi-Layer Perceptron (MLP) classifiers were optimized and tested for classification. Among utilized models, Random Forest performed best with higher F1-scores compared to others (0.772), marginally better than the closest MLP (0.745) and LightGBM (0.725). The number of comments, priority, and severity features alongside main text features made the highest contribution towards prediction. Experiment confirms that combining structured metadata and text information improves classification accuracy and provides actionable feedback to allow teams to maximize prioritization and bug-fixing allocation.

Keywords
Machine learning, predictive modeling, classification, bug resolution time, software maintenance, feature importance

1. INTRODUCTION
Bug resolution time takes up a considerable amount of time in the software development process, and this time directly affects the release time and planning. Accurately predicting bug resolution time remains challenging due to the heterogeneous nature of bug reports, which combine structured metadata with unstructured textual descriptions and exhibit complex lifecycle dynamics.

Over the past decades, researchers have attempted numerous alternatives to solve this issue, ranging from statistical models and conventional machine learning to deep learning methodologies. Early studies primarily used simple metadata such as severity, priority, and assignee information, and obtained good but limited predictive performance [1, 2]. Subsequent work incorporated textual characteristics extracted from bug descriptions and comments using natural language processing techniques [3, 4], while more recent studies also attempted sequence models to capture the dynamic evolution of bug lifecycles [5, 6]. Despite these advances, several critical gaps remain: (1) limited exploration of semantic content from bug descriptions combined with structured metadata, (2) poor performance in predicting long-duration bugs that are most critical for project planning, and (3) limited model interpretability that reduces practical adoption by software teams.

This study focuses on bug resolution time prediction through a classification-based approach that categorizes bugs as fast or slow resolving. Using publicly available bug tracking datasets, two feature categories are investigated: structured metadata (severity, priority, comment count) and textual content from bug summaries extracted using TF-IDF representations and sentence transformers. Five machine learning algorithms (RandomForest, LogisticRegression, LightGBM, SGD Classifier, and MLP) are trained and evaluated, emphasizing both prediction accuracy and model interpretability through systematic feature importance analysis. This research is guided by three primary questions. First, the effectiveness of machine learning algorithms in predicting bug resolution speed using historical bug report data is investigated. Second, the analysis examines which features—metadata versus textual content—are most predictive of bug resolution time, and how their relative contributions vary across different model architectures. Third, classification algorithms are compared in terms of accuracy, interpretability, and practical applicability for bug resolution time prediction, considering both performance metrics and real-world deployment feasibility.

This research contributes to bug resolution time prediction by providing a comparison of machine learning approaches with emphasis on interpretability. The findings aim to deliver usable findings for project managers and development teams, enabling better resource allocation and project planning decisions through more transparent and reliable prediction models.

2. LITERATURE REVIEW
Bug resolution time prediction is a well-studied topic in software engineering due to its importance for effective project management and prioritization. While various approaches ranging from traditional machine learning and statistical methods to deep networks and ensemble models have brought improvements, there are still challenges, such as feature selection and limited cross-project generalizability. This chapter addresses the most relevant works, which are thematically categorized into four categories: traditional machine learning approaches, sequential and temporal modeling, text-based and deep learning methods.

2.1 Classical Machine Learning Approaches
Early research applied relatively primitive machine learning models to forecast bug fix time. Abdelmoez et al. [2] developed a Naïve Bayes classifier using bug reports from four large open-source systems: Eclipse JDT, Mozilla Firefox, Gnome GStreamer, and Gnome Evolution. Fix time was discretized into quartiles to create categorical classes such as “very fast”

and "very slow." Their 17 metadata attributes included severity, priority, assignee, and number of comments. Findings showed that while recall for "very fast" bugs was poor (e.g., 0.20 for Q1 in Eclipse JDT), the model was more accurate in classifying slower bugs (recall ~0.90) and showing the efficacy of simple classifiers when paired with prudent feature engineering. The poor recall for fast bugs (0.20) suggests that metadata alone may be insufficient for capturing the complexity of rapid bug resolution patterns.

Bhattacharya and Neamtiu [1] conducted a large-scale analysis using univariate and multivariate regression on 512,474 bug reports from prominent open-source projects. Their models achieved limited predictive power (R^2 ranging from 30% to 49%), and notably, features such as bug reporter reputation showed minimal impact on resolution time. These findings point to the inadequacy of relying solely on conventional metadata features and underscore the need for more sophisticated feature engineering approaches to improve prediction accuracy.

2.2 Temporal and Sequential Modeling

Beyond static features, researchers began exploring temporal and sequential dynamics in bug lifecycles. Habayeb et al. [5] employed Hidden Markov Models (HMMs) to model developer activity sequences, including assignments, comments, and reviews in Firefox bug reports. Using median resolution time as the threshold for "fast" versus "slow" classification, their HMM approach achieved approximately 10% improvement in prediction accuracy compared to frequency-based classifiers. This work demonstrated the value of modeling bug resolution as a dynamic process rather than a static snapshot.

Zhang, Gong, and Versteeg [7] applied temporal and stochastic methods to commercial software development projects. Bug state transitions were represented using a Markov model, Monte Carlo simulations to forecast the overall fixing effort, and a k-nearest neighbors classifier with ad hoc distance measures for distinguishing quick and slow fixes. Their evaluation on three large CA Technologies projects had low error in prediction (mean relative error 3-6%) and good classification accuracy (average F-measure of about 72%).

Akbarinasaji et al. [8] replicated the Markov and Monte Carlo approach of Zhang et al. [7] on Mozilla Bugzilla data. They confirmed the validity of such processes but also noted variation in fix time behavior between backlog and project policy, illustrating the benefit of replication in validating temporal models.

2.3 Text-Based Approaches

The increased volume of textual comments and bug reports has promoted techniques that leverage natural language processing via standard ML methods. TF-IDF and PCA were employed by Ardimento et al. [3] on the description of bugs from four open-source software projects, evaluating regressors such as support vector machines, random forests, and M5P. While achieving moderate classification accuracy (0.74-0.78), their RMSE of 123-152 hours indicates substantial prediction variance, particularly compared to the 3-6% relative error reported by Zhang et al. [7] using temporal modeling. However, "slow" bug recall remained low (~0.01-0.12), highlighting continuing class imbalance problems. This observation serves to support the need to explore complementary feature types like metadata, text, and temporal dynamics in an integrated framework.

2.4 Deep Learning Methods

Deep learning was utilized to derive semantic and sequential data from bug reports. Sepahvand et al. [4] presented DeepLSTMPred, which used word embeddings and LSTM networks to capture semantic and sequential data from Mozilla project bug reports. Their approach achieved 15-20% higher performance on all measures compared to HMM, showing how deep learning could outperform when it comes to handling textual and temporal relationships simultaneously.

Du et al. [6] had constructed a dual-sequence LSTM model that incorporated event order and timestamp features, improving upon conventional sequence models such as HMMs by up to 10% F1-score. Özkan et al. [9] applied neural network regression to ONAP and ONOS JIRA issue data. Their features were workflow transitions, usernames of reporters and assignees, and priority, with a normalized median error of ~20%.

The literature reveals a clear evolutionary progression in bug resolution time prediction, advancing from classical machine learning through temporal modeling to sophisticated deep learning approaches. Each generation has addressed specific limitations of its predecessors while introducing new capabilities.

Traditional metadata-based models established foundational approaches and provided interpretable baselines, but demonstrated limited predictive power ($R^2 < 50\%$) and severe class imbalance issues. Temporal and sequential models significantly improved accuracy by modeling bug lifecycles as dynamic processes, achieving notable reductions in prediction error (3-6% relative error vs. 123-152 hour RMSE).

Deep learning approaches have further improved performance by enabling richer semantic representations and joint modeling of textual and temporal information, reporting gains of approximately 15-20% over previous methods. However, several challenges remain unresolved. Prior studies consistently report severe class imbalance, particularly for long-duration bugs, where recall for slow-fixing cases remains extremely low (0.01-0.12). In addition, limited cross-project generalizability continues to constrain practical deployment, and the lack of interpretability in more complex models hinders their adoption by software development teams.

Collectively, existing research suggests steady progress in bug resolution time prediction, while showing that improvements in prediction have often come at the cost of model interpretability. Future research should prioritize addressing the mentioned issues while leveraging the complementary strengths of metadata, textual, and temporal features in integrated, interpretable frameworks.

3. MATERIALS AND METHODOLOGY

3.1 Dataset Characteristics

The dataset employed for the research was obtained from BugsRepo [10], which is a sanitized collection of bug reports obtained from the Bugzilla issue tracking database. The dataset collates bug reports from different Mozilla projects, and hence it is an appropriate benchmark dataset for software engineering research.

The original dataset consists of 225,196 bug reports collected from Bugzilla. The reports span a time period from January 2021 to November 2024, covering several years of software

development and maintenance activity. Unlike pre-filtered sections of the dataset, the raw dataset preserves the full distribution of reports, heterogeneous quality of metadata and more textual data. This set ensures that prediction models are tested in the same noisy and incomplete environment that practitioners would typically encounter in real life. The dataset contains the following information:

- Metadata fields: priority, severity, status, resolution, comment count, and assignee.
- Lifecycle information: bug creation timestamp, modification timestamps, closure timestamp, enabling calculation of resolution time.
- Contributor information: reporters, assignees, and developers who have commented on the bug report.
- Textual content: brief summaries and full descriptions of issues reported.

An initial inspection of data revealed missing values in several attributes. The severity field contains missing values in 15,716 reports (6.98%), while the "cf_last_resolved" timestamp is missing in 47,578 reports (21.13%). Other core metadata attributes, such as summary and status, have no missing values. These missing data patterns informed subsequent preprocessing and handling strategies described later in this section.

3.2 Data Analysis and Preprocessing

The analysis and preprocessing steps were designed based on the observed characteristics of the dataset. The entire training and testing process took a systematic approach, as illustrated in Figure 1.

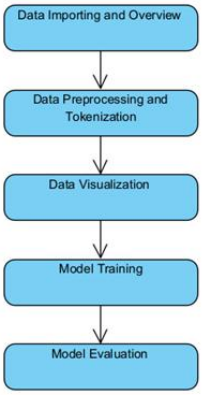

Fig 1: Bug Resolution Prediction Pipeline

Of the initial group of bug reports, only those with resolution equal to 'resolved', and status equal to 'fixed' or 'verified' were kept in order for meaningful calculation of resolution time. This filtering step reduced the dataset to 95,567 bug reports. The resolution time, which is the target variable, was directly acquired from the subtraction of bug closing and creation timestamps.

The exploratory analysis of the distribution of resolution time showed a highly right-skewed pattern: most bugs get resolved quickly, but there is a long tail where bugs require much longer fixing times. Figure 2 shows this distribution. Strong skew and the presence of extreme values motivated framing the problem as a classification task rather than direct regression, which would be sensitive to outliers.

To prevent unrealistically long fixing times from disproportionately influencing model training, extreme outliers were removed by excluding bugs with resolution times above the 99th percentile of the filtered distribution. This step removed only the longest-running bugs in the tail of the distribution while preserving the overall structure of the data and improving model stability.

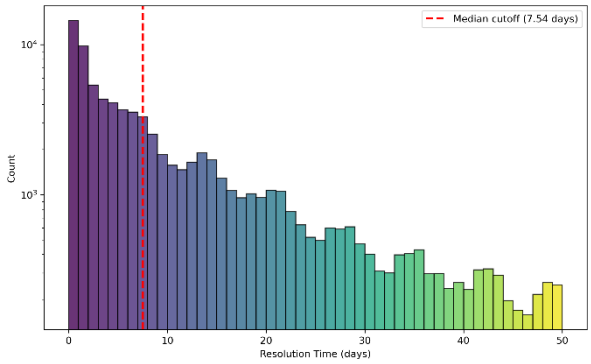

Fig 2: Number of Bugs per Resolution Time (0-50 Days)

The median resolution time for filtered data was 7.5 days, as indicated by the reference line in Figure 2. This value served as a threshold for dividing bugs into fast (< 7.5 days) and slow (> 7.5 days) resolution categories. The median-based split was chosen to ensure that classes were well-balanced while maintaining robustness against extreme values. Figure 3 presents the resulting class distribution, which balances both classes with around 47,300 bug reports and thus mitigates class imbalance in this respect.

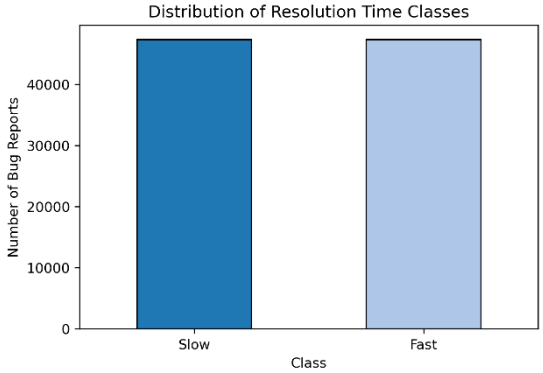

Fig 3: Bug Resolution Class Distribution (Fast vs Slow)

Missing data patterns identified in Section 3.1 informed preprocessing decisions. Placeholder values such as "--" in the "severity" and "priority" columns were treated as "Unknown". Severity levels (e.g., "critical"," major") were converted to their respective numeric levels. Low-cardinality categorical columns (e.g., severity, priority) were label-encoded, and high-cardinality categorical columns (product, component) had sparse categories aggregated into an "Other" category and then encoded into binary indicator variables via one-hot encoding. Numerical features (number of comments) were log-transformed to eliminate heavy skew.

Text data from bug report summaries was converted to lowercase, stopwords were removed, and lemmatization was applied at the first step. For the MLP model that works well with dense feature representations, bug summaries were encoded using sentence embeddings generated by a pre-trained "all-MiniLM-L6-v2" model from the Sentence-Transformers library [11]. This model is based on the MiniLM architecture

[12], and captures more semantic information than word frequency. The embedding representation was used for the Random Forest, LightGBM, and Multilayer Perceptron models. For simpler models, the text was vectorized with a Term Frequency-Inverse Document Frequency (TF-IDF) representation. Vectorizer was configured to a maximum vocabulary size of 10,000, and both unigrams and bigrams were utilized to capture short phrases meaningful in bug reports.

These phases generated two primary collections of features: structured metadata and vector representations of text summaries.

3.3 Model Training

Several machine learning algorithms were used to compare predictive performance on classification tasks for bug resolution time. The final set of models included Random Forest, LightGBM, Logistic Regression, SGD Classifier, and a Multilayer Perceptron (MLP) implemented in PyTorch (Table 1). These models were selected to cover a wide methodological spectrum - from linear classifiers optimized for high-dimensional text data to ensemble-based models that are robust to noise and heterogeneous features, and neural network approaches that are capable of capturing nonlinear relationships and complex feature interactions. This selection will offer insight into the robustness of different algorithmic approaches to this predictive task.

Table 1. Feature types used for each evaluated model

Model	Features
Logistic Regression	TF-IDF text representations (report summary), numeric (number of comments, text length), and categorical metadata (severity, priority, product, component)
LightGBM	TF-IDF text representations, numeric and categorical metadata
Random Forest	TF-IDF text representations, numeric and categorical metadata
SGD Classifier	TF-IDF text representations, numeric and categorical metadata
MLP	Sentence embeddings, numeric and categorical metadata
LR (Text Only)	TF-IDF text representations
LR (Num + Cat)	Numeric and categorical metadata

To ensure fairness and comparability, all models were trained and tested using a consistent setup with five-fold cross-validation. This approach enables estimation of the performance variability across folds and models' performance on unseen data.

Logistic Regression was run in multi-class mode and set to a maximum of 1,000 iterations to guarantee convergence. It was used here as a linear baseline model for interpretability and ease of use with high-dimensional TF-IDF features. The SGD Classifier was also trained on TF-IDF sparse vectors combined with numerical and categorical attributes, adding another linear model that can optimize efficiently on large feature spaces by stochastic gradient descent.

In contrast, Random Forest was trained with 200 estimators, leveraging its ensemble structure and robustness against noisy and heterogeneous features [13]. LightGBM was initialized with 600 estimators, benefiting from its gradient-boosting framework optimized for efficiency and strong performance on structured and mixed-feature inputs [14].

The Multilayer Perceptron model implemented in PyTorch was trained for 80 epochs using cross-entropy loss and the Adam optimizer. As opposed to previous models, it used dense semantic text embeddings combined with numerical and categorical features. Dense embeddings capture semantic relationships in a compact representation and are favorable for non-linear models such as neural networks [15, 16]. The trained MLP consisted of fully connected layers with non-linear activations and dropouts. Two MLP variants were evaluated: one trained only on text embeddings and another combining embeddings with additional features, enabling an assessment of the contribution of structured metadata to predictive performance.

Two baseline classifiers were trained using Logistic Regression. The first one uses only textual data from reports, and the second one uses a combination of numerical and categorical data. These baselines are used to exhibit performance gains in the main models.

Along with cross-validation, a separate temporal analysis was performed to simulate a real-world scenario. In this approach, the models were trained on bugs reported in 2021-2022, tested on data from 2023, and further validated on 2024 data. This chronological setup enabled assessment of model performance on future, unseen data.

3.4 Model Evaluation

Model performance was estimated using accuracy, precision, recall, Matthews Correlation Coefficient (MCC), and F1-score with macro-averaging across classes to mitigate the effect of variation across classes. The reported results give the mean and standard deviation across five folds, which effectively shows performance stability.

Receiver Operating Characteristic (ROC) curves and the Area Under the ROC Curve (AUC) were used to evaluate how well each model separates the two classes across different decision thresholds, where a threshold defines the cutoff used to assign predictions to a class. In addition, confusion matrices were generated to provide an in-depth perspective into what characterizes common misclassification patterns. Matthews Correlation Coefficient (MCC) was computed as a balanced measure that takes into account all elements of the confusion matrix. Precision estimates the proportion of the number of cases correctly predicted to be in a class out of the total cases predicted to be that class, showing how well the model makes "reliable" predictions. Recall estimates the proportion of the number of true cases of a class correctly identified, showing how well the model can pick out all relevant instances. F1-score is the harmonic mean between recall and precision and gives a balanced value that considers both the false positives and the false negatives. Accuracy complements these metrics by showing the overall fraction of correct predictions [17]. Baseline models were evaluated using the same metrics to ensure a fair comparison. Ultimately, feature importance analysis was performed, from which the most significant features were extracted. These analyses provided interpretability by delivering structured attributes and textual features that contributed the most towards predictions [18].

4. RESULTS

This section presents the results of bug resolution time classification. Five machine learning models were trained and

evaluated to classify bugs into fast or slow. Model performance was measured using five-fold cross-validation and multiple baselines in terms of accuracy, precision, recall, F1-score, MCC, and ROC-AUC, with mean values and standard deviation used to capture stability.

4.1 Overall Model Performance

Results in Table 2 show the cross-validation results for the tested models, as well as two baseline classifiers trained on the respective feature sets. For the classifiers trained on the combined feature set, the Random Forest classifier has the best performance, obtaining a macro F1-score of 0.772 ± 0.001 and accuracy of 0.773 ± 0.001. The Multilayer Perceptron and LightGBM classifiers have slightly lower but comparable performance, obtaining macro F1-scores of 0.745 ± 0.005 and 0.725 ± 0.002, respectively. Linear classifiers, such as the Logistic Regression and the SGD Classifier, have lower but stable performance.

Table 2. Overall Model Performance Comparison

Model	Accuracy (mean ± std)	Precision (macro)	Recall (macro)	F1-score (macro)
Logistic Regression	0.701 ± 0.003	0.701 ± 0.003	0.701 ± 0.003	0.701 ± 0.003
LightGBM	0.725 ± 0.002	0.726 ± 0.002	0.725 ± 0.002	0.725 ± 0.002
Random Forest	**0.773 ± 0.001**	**0.773 ± 0.001**	**0.773 ± 0.001**	**0.772 ± 0.001**
SGD Classifier	0.699 ± 0.001	0.700 ± 0.001	0.699 ± 0.001	0.699 ± 0.001
MLP	0.745 ± 0.005	0.745 ± 0.005	0.745 ± 0.005	0.745 ± 0.005
LR (Text Only)	0.654 ± 0.001	0.654 ± 0.001	0.654 ± 0.001	0.654 ± 0.002
LR (Num + Cat Only)	0.643 ± 0.003	0.643 ± 0.003	0.643 ± 0.003	0.643 ± 0.003

The performance of the baseline classifiers is well below that of the models using the whole set of features. The Text Only Logistic Regression baseline reaches a macro F1-score of 0.654 ± 0.002, while the model trained on the numerical and categorical metadata only reaches 0.643 ± 0.003. These results suggest that the predictive information contained in the text data and the metadata is complementary.

In all tested models, the standard deviations of the observed values remain small (0.001-0.006), which indicates that the models are less sensitive to the data splitting. The results suggest that the results presented are reliable and represent the generalization performance.

The Matthews Correlation Coefficient score follows the trend seen in other metrics. Random Forest achieved the highest score at 0.546 ± 0.002, followed by MLP at 0.494 ± 0.003 and LightGBM at 0.451 ± 0.005. Logistic Regression achieved the score of 0.403 ± 0.006. Baseline models consistently had lower MCC scores of 0.308 and 0.286.

Models were further evaluated using a separate temporal split scenario in order to assess robustness. Due to the increased difficulty of forecasting future data distributions, overall performance declined when compared to cross-validation results. Random Forest remained the top-performing model with the highest F1-scores on the 2023 set (0.702) and 2024 set (0.629). LightGBM followed closely with scores of 0.700 and 0.626. Compared to Random Forest, LightGBM had a smaller performance drop from cross-validation to the temporal test split, indicating better stability. Conversely, MLP showed the largest decline in performance, dropping from a cross-validation F1-score of 0.745 to 0.661 on the 2023 set and 0.605 on the 2024 set, suggesting lower robustness to temporal distribution shifts.

To evaluate model discriminative ability independently of classification thresholds, ROC curves were plotted. These are shown in Figure 4. The model showing the maximum AUC is the Random Forest model with an AUC of 0.855, while the second-best result is delivered by the Multilayer Perceptron model (0.819), followed by the LightGBM model (0.765). The models utilizing the complete set of features have been performing better than the baseline models over the entire range of thresholds.

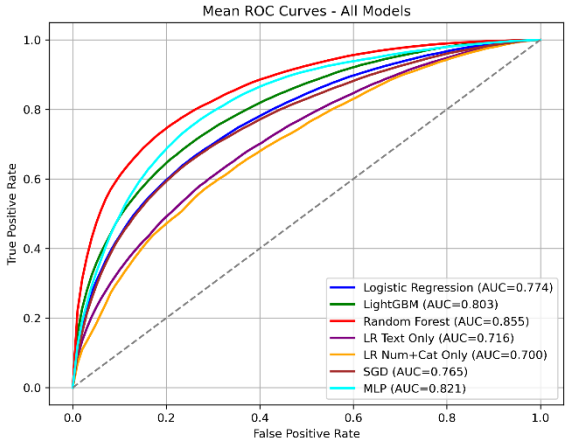

Fig 4: ROC Curves for All Trained Models

4.2 Class Specific Performance

In order to gain a better understanding of how models perform, it was tested how well each classifier performed on each class separately. Class-specific F1-scores are summarized in Table 3. Noticeably, trained models perform slightly better for slow-resolving bugs than for fast-resolving bugs, which implies that slow bugs have a clearer pattern in their combined features.

Table 3. Class-specific F1 scores

Model	F1 Fast	F1 Slow
Logistic Regression	0.697 ± 0.003	0.705 ± 0.003
LightGBM	0.719 ± 0.003	0.731 ± 0.002
Random Forest	0.766 ± 0.002	0.779 ± 0.001
SGD Classifier	0.690 ± 0.005	0.708 ± 0.004
MLP	0.744 ± 0.005	0.746 ± 0.005
LR (Text Only)	0.648 ± 0.003	0.660 ± 0.002
LR (Num + Cat Only)	0.642 ± 0.003	0.643 ± 0.004

The highest score is again achieved by Random Forest Classifier, 0.779 ± 0.001 for "slow" class and 0.766 ± 0.002 for bugs classified as "fast".

In the confusion matrix in Figure 5, the main errors made by the Random Forest algorithm appear to be in the fast-resolving bugs, where the algorithm mistakenly categorizes them as slow.

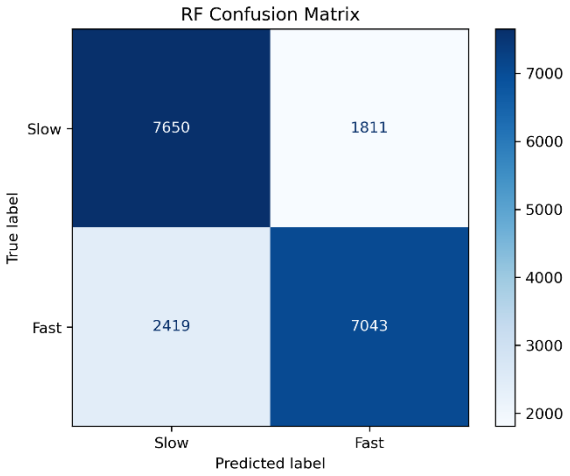

Fig 5: Random Forest Confusion Matrix

This indicates that the characteristics of the slow bugs, such as those including low priority, many comments, or lengthy descriptions, can be distinguished better by the algorithm. The fast bugs may have some common characteristics that make them hard to distinguish. Such trends can also be seen in other classifiers, such as LightGBM and Multilayer Perceptron, although the error counts will differ slightly. Linear classifiers have lower class-specific F1-scores, likely due to their limited capacity to model non-linear relationships between textual and structured features.

4.3 Feature Importance and Interpretability

To make the model more interpretable and identify the most contributing attributes for text and non-text attributes in predicting bug resolution time, a feature importance analysis was conducted on textual and structural metadata using Logistic Regression and the Random Forest model. The LR gives interpretable coefficients, while RF takes into account the nonlinear relations and the relevance of features.

4.3.1 Textual Feature Importance

Textual features extracted from bug summaries and titles are important for distinguishing fast-resolving bugs from slow-resolving ones. In the LR model, terms like "meta" (LR coef ≈ -7.0), "wpt failure" (≈ -3.1), "implement" (≈ -2.8), and "chatzilla" (≈ -3.0) are closely linked to the slow-resolving class. This suggests that issues related to infrastructure changes, testing frameworks, or tasks that require significant implementation usually take longer to fix. In contrast, terms like "perma" (≈ +5.5), "update pdf" (≈ +2.5), and "add" (≈ +2.4) are more often connected to quick resolutions. This indicates that these terms relate to maintenance tasks or specific changes. The RF importance analysis confirms these findings. Several meaningful tokens, such as "wpt", "sync", "remove", "update", and "test", emerge as some of the most important features (RF importance ≈ 0.005). Absolute importance values are low, given the high dimensionality of the used TF-IDF. This shows that ensemble models capture the interactions between textual meaning and structured metadata. Based on this, it can be confirmed that semantic content is a key indicator of how quickly a bug can be resolved.

4.3.2 Structured Feature Importance

Structured metadata features also significantly impact predictive performance. In both the LR and RF models, "num_comments_log" stands out as a key predictor (RF importance ≈ 0.06; LR coef ≈ −0.73). This shows that bugs needing a lot of discussion are likely to take longer to resolvePriority and severity influence prediction by showing how urgent the workflow is and how much impact it has. Additionally, categorical features related to product and component groups have moderate and consistent effects. For example, component categories like "Administration" (LR coef ≈ +1.34) are associated with faster resolutions. In contrast, some product categories, such as "Data Platform and Tools" (≈ −1.11), relate to longer fixing times. This indicates that subsystem complexity and organizational factors affect the speed of issue resolution. The RF model also emphasizes the significance of basic text properties. For instance, "text_length" (RF ≈ 0.02) suggests that longer and more detailed bug descriptions can lead to slower resolutions.

4.4 Discussion

The models reviewed show that bug resolution speed can be predicted with reasonable accuracy using historical bug reports. Ensemble and neural models perform the best, providing consistent cross-validation results and clear improvements over baseline classifiers. This suggests that machine learning methods work well for this task in the evaluated setting. The observed performance levels are comparable to, and in some cases better than, those reported in earlier studies that focused only on metadata or textual features.

With respect to feature contributions, both textual and structured features play an important role in prediction. Textual features are helpful in understanding the complexity of the task and the type of task. The structured metadata provides information about the workload, urgency, and the context of the organization. Notably, there is no category of features that is universally best suited to all models. Performance improvements are achieved through their combination, indicating that the employed feature types provide complementary information for bug resolution time prediction.

From a modeling perspective, nonlinear models and ensemble models show a clear edge in predictive capability over linear models, especially when using more sophisticated feature sets. However, linear models and Random Forest models provide a clear interpretability advantage that allows for easy examination of the relevant features. Out of the models considered, the Random Forest model provides a good balance between interpretability and predictive capability, making it suitable for practical deployment in bug resolution time prediction tasks.

The temporal split results further confirm that ensemble models generalize better to future data, while simpler linear models are more sensitive to temporal distribution shifts. The performance decay observed in this scenario suggests that periodic model recalibration may be necessary as software defect characteristics evolve.

In general, these results are in line with existing work suggesting the addition of textual data to structured metadata improves the prediction of bug resolution time, and that nonlinear models gain the most from richer representations.

5. CONCLUSION

This study investigated the use of machine learning for predicting bug resolution time by classifying reported bugs as either fast or slow. A large bug report dataset was preprocessed and examined, including both structured metadata and text-derived features from bug summaries. Several machine learning models were compared, including Logistic Regression, SGD Classifier, Random Forest, LightGBM, and a Multilayer Perceptron, representing different approaches to handling textual and heterogeneous data.

These experiments demonstrated that the Random Forest had the strongest overall performance, with the Multilayer Perceptron closely behind. In general, the models performed well in distinguishing between fast and slow bug resolutions, although slightly better accuracy and recall were observed for the slow cases. The feature importance analysis of the RF and LR models confirmed that comment count, priority, and severity are influential determinants of resolution time, together with words that highlight specific actions. The main contributions of this work are a comparative evaluation of several prediction models and empirical evidence that engineered metadata features improve the classification performance in combination with semantic text representations.

Despite its contributions, this study has several limitations. Some bugs near the fast/slow boundary remain hard to classify consistently, and the findings apply most directly to projects based on Bugzilla. Performance could potentially be improved, and the findings could be generalized to other issue tracking systems by incorporating temporal or sequence-aware modeling, adopting deeper learning architectures, or applying explainability techniques. The paper shows that it is possible to predict bug-fixing times with good accuracy based on metadata, text features, and machine learning algorithms, supporting planning, prioritization, and resource allocation in software maintenance.

6. REFERENCES

[1] Bhattacharya, P. and Neamtiu, I. 2011. Bug-fix time prediction models: can we do better? In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11). Association for Computing Machinery. https://doi.org/10.1145/1985441.1985472

[2] Abdelmoez, W., Kholief, M., and Elsalmy, F. M. 2012. Bug fix-time prediction model using naïve Bayes classifier. In Proceedings of the 2012 22nd International Conference on Computer Theory and Applications (ICCTA). Alexandria, Egypt. https://doi.org/10.1109/ICCTA.2012.6523564

[3] Ardimento, P., Boffoli, N., and Mele, C. 2020. A text-based regression approach to predict bug-fix time. https://doi.org/10.1007/978-3-030-36617-9_5

[4] Sepahvand, R., Akbari, R., and Hashemi, S. 2020. Predicting the bug fixing time using word embedding and deep LSTM. IET Software, 14. https://doi.org/10.1049/iet-sen.2019.0260

[5] Habayeb, M., Murtaza, S. S., Miranskyy, A., and Bener, A. B. 2018. On the use of Hidden Markov Model to predict the time to fix bugs. IEEE Transactions on Software Engineering, 44, 12, 1224–1244. https://doi.org/10.1109/TSE.2017.2757480

[6] Du, J., Ren, X., Li, H., Jiang, F., and Yu, X. 2022. Prediction of bug-fixing time based on distinguishable sequences fusion in open source software. Journal of Software: Evolution and Process, 35. https://doi.org/10.1002/smr.2443

[7] Zhang, H., Gong, L., and Versteeg, S. 2013. Predicting bug-fixing time: An empirical study of commercial software projects. In Proceedings of the 2013 35th International Conference on Software Engineering (ICSE). San Francisco, CA, USA. https://doi.org/10.1109/ICSE.2013.6606654

[8] Akbarinasaji, S., Caglayan, B., and Bener, A. 2018. Predicting bug-fixing time: A replication study using an open source software project. Journal of Systems and Software, 136, 173–186. https://doi.org/10.1016/j.jss.2017.02.021

[9] Ozkan, H. Y., Heegaard, P. E., Kellerer, W., and Mas-Machuca, C. 2024. Bug analysis towards bug resolution time prediction. arXiv. https://doi.org/10.48550/arXiv.2407.21241

[10] Acharya, J. and Ginde, G. 2025. BugsRepo: A comprehensive curated dataset of bug reports, comments, and contributors information from Bugzilla. arXiv. https://doi.org/10.48550/arXiv.2504.18806

[11] Reimers, N. and Gurevych, I. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. arXiv. https://arxiv.org/abs/1908.10084

[12] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. 2020. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. arXiv. https://arxiv.org/abs/2002.10957

[13] Breiman, L. 2001. Random forests. Machine Learning, 45, 1, 5–32. https://doi.org/10.1023/A:1010933404324

[14] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. 2017. LightGBM: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems 30.

[15] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. 2019. PyTorch: An imperative style, high-performance deep learning library. arXiv. https://arxiv.org/abs/1912.01703

[16] Popescu, M.-C., Balas, V. E., Perescu-Popescu, L., and Mastorakis, N. 2009. Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems, 8, 7.

[17] Powers, D. M. W. 2011. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. Journal of Machine Learning Technologies, 2, 1, 37–63.

[18] Molnar, C. 2020. Interpretable machine learning: A guide for making black box models explainable. https://christophm.github.io/interpretable-ml-book/

[19] Scikit-learn. n.d. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/

[20]