ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 | Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix: https://doi.org/10.33050/icit.v12i2.4385

SVM-Based Sentiment Analysis for Bug Tracking Priority Scale

Lutfi Hibatullah Dwi Putra[*1], Deden Rustiana[2], Nina Rahayu[3]
[1,2,3]University of Raharja, Tangerang, Indonesia
*Corresponding author: lutfi.hibatullah@raharja.info

Article Information:
Received: 11 May 2026
Revised: 10 June 206
Accepted: 08 July 2026
DOI: 10.33050/icit.v12i2.4385

ABSTRACT
The rapid expansion of mobile applications in Indonesia creates a massive density of user reviews, making manual categorization of technical bugs highly inefficient for development teams. This research develops an automated bug tracking system powered by sentiment analysis to address these constraints. Utilizing a dataset of 1,200 reviews extracted from the Google Play Store (2023-2024), the study compares the Support Vector Machine (SVM) algorithm with a Radial Basis Function (RBF) kernel against the Naive Bayes classifier. The methodology encompasses automated scraping, text preprocessing via the Sastrawi stemming library, and TF-IDF feature weighting to handle semantic ambiguities in informal Indonesian text. Evaluation results demonstrate that SVM achieves a superior accuracy of 86.25% and an F1-score of 0.86, significantly outperforming Naive Bayes which stands at 77.50%. McNemar's test yields a p-value of 0.006, statistically validating SVM's effectiveness in reducing False Negatives. While highly successful in rule-based priority scaling, the architecture automatically classified the actionable technical bug reports into 150 Critical, 200 Major, and 50 Minor urgency categories. Operationally, the automated system processes data 99.8% faster than manual methods, cutting down the identification period from 12 hours to just 85 seconds. These findings imply that embedding SVM into bug tracking architectures facilitates real-time mitigation of critical stability issues. While highly successful, future studies should focus on overcoming linguistic limitations regarding sarcasm detection through advanced deep sequential processing.

Keywords—Sentiment Analysis, Support Vector Machines, Bug Tracking System, Natural Language Processing, TF-IDF

1. INTRODUCTION

The expansion of the mobile app ecosystem in Indonesia continues to show exponential growth, with the active internet user base officially reaching 221.5 million users according to the latest empirical data from the Indonesian Internet Service Providers Association (APJII) [1]. This surge in penetration is directly proportional to the volume of user feedback on distribution platforms like the Google Play Store. While these reviews are a vital asset for software evolution, the massive data density often exceeds the cognitive capacity of development teams to categorize them conventionally. Empirical findings indicate that consumer loyalty in the digital sector risks a decline of up to 89% if technical issues are not addressed promptly through a rapid response to user complaints [2].

This situation is exacerbated by industry data from the Global Software Quality Report (2023), which shows that approximately 25% of users will permanently abandon an app after experiencing just one technical failure (crash) that is not quickly addressed [3]. Furthermore, a Statista Intelligence report (2024) highlighted that the accumulation of negative reviews related

to functionality directly impacts an app's visibility ranking on Search Engine Optimization (SEO) platforms by 40%, ultimately significantly reducing the number of new downloads [4]. The fundamental constraints of traditional bug tracking systems lie in their reactive nature and the communication gap between subjective user complaints on the front-end and technical management on the back-end [5].

Amidst increasingly competitive markets, development teams are being urged to shift from time-consuming manual Quality Assurance (QA) models to AI-driven QA systems capable of continuous early detection [6]. The integration of sentiment analysis based on the Support Vector Machine (SVM) algorithm offers a strategic solution to bridge this gap. SVM is known for its superior performance in handling high-dimensional text data with a stable level of precision, particularly in Indonesian language reviews, which are rife with semantic ambiguity and informal language [7].

The primary novelty of this research lies in the design of an automated system architecture that seamlessly maps machine learning sentiment predictions directly into a real-time rule-based priority ranking. Unlike previous works that stop at sentiment classification, this approach dynamically quantifies user text into an engineered triage weight. Based on these challenges, this research focuses on developing a system capable of automatically transforming user review narratives into a measurable priority scale. By integrating SVM-based sentiment classification into the Bug Tracking System mechanism, development teams can mitigate critical issues in real time. This research is expected to not only improve the operational efficiency of software development but also contribute to proactive application quality maintenance to maintain user trust in the digital age.

2. LITERATURE REVIEW

Various previous literature reinforces the urgency of using sentiment analysis in the Software Analytics ecosystem. Integrating sentiment analysis into the software development life cycle (SDLC) has been shown to reduce complaint identification time by up to 60% compared to manual methods [8]. Recent research shows that real-time sentiment mining demonstrates significant time-efficiency improvements in supporting agile software development cycles [9]. Furthermore, projections for the global app market in 2025 predict a surge in review volume, requiring automated feedback analysis to maintain product relevance in emerging markets [10].

However, implementation in Indonesian-language reviews faces challenges due to linguistic complexity, which is rich in slang and complex morphology. In this regard, utilizing the Sastrawi library is a vital component in the preprocessing stage to standardize words through stemming, which has been reported to increase text processing effectiveness by up to 15% [11]. In addition to technical efficiency, proactive bug management through automation plays a crucial role in reducing technical debt and reducing app churn rates by up to 34% [12].

In terms of classification algorithms, a literature review shows that while Naive Bayes excels in execution speed on high-dimensional data [5], the Support Vector Machine (SVM) algorithm with a Radial Basis Function (RBF) kernel offers better stability in handling short, irregular reviews [7]. A study by Jaya et al. (2024) confirmed this performance, with SVM leading with an accuracy of 84.15%, surpassing Naive Bayes at 77.64% for software review classification [13]. Although Linear Kernels are standard for sparse, high-dimensional text vectors under TF-IDF representation, the RBF kernel was explicitly selected in this research to accommodate highly non-linear decision boundaries introduced by mixed emotional text, typos, and heavy colloquial noise characteristic of localized Indonesian software feedback where standard linear boundaries generate excessive classification errors. By converting the results of this analysis into urgency levels (High, Medium, Low), the development team can prioritize fixes more accurately based on user emotional validation [8]. Based on this foundation, this study aims to optimize the reliability of the bug tracking system through precise text processing and real-time automatic mapping.

ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 |
Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix:
https://doi.org/10.33050/icit.v12i2.4385

3. METHODOLOGY

3.1. Research Framework

This research uses a quantitative approach with textual data experiments. The research structure is divided into data preparation, artificial intelligence-based modeling, and system integration, which is illustrated comprehensively in Figure 1. This approach is designed to support the transition from traditional quality management to more efficient predictive maintenance [6]. The entire flow is constructed linearly to ensure scientific accountability from the extraction stage to the visualization stage.

Figure 1. Research framework and system architecture flowchart

3.2. Automated Data Acquisition Mechanism

Primary data was collected through an automated scraping mechanism on the Google Play Store using the google-play-scraper library in Python. To ensure high generalizability and experimental reproducibility, the targeted data objects were sourced exclusively from user reviews of three leading Indonesian mobile commerce (m-commerce) applications within the digital retail domain during the 2023-2024 period. The specific 2023–2024 scraping timeframe was deliberately chosen due to a series of major UI/UX overhauls and system migration phases executed by the target applications during this period, which historically triggered an unprecedented surge in user technical complaints and regressions. The selection criteria for these applications included: (1) a minimum download threshold of 10 million users on the Google Play Store, (2) an active daily user base, and (3) a high daily volume of textual feedback filled with localized linguistics. Extracted attributes included review content, rating scores, timestamps, and app versions to track the relevance of technical issues [14].

3.3. Data Labeling Process

The 1,200 reviews collected through the acquisition stage were processed through manual labeling by the researchers to determine the ground truth or model truth labels. The labeling process was carried out consistently, referring to pre-defined sentiment classification guidelines, where reviews were categorized into three initial classes: Positive, Negative, and Neutral.

To focus the model on technical bug detection, it was simplified to a binary classification (Bug vs. Non-Bug). In this stage, the Neutral and Positive classes were combined into a Non-Bug category. This was done because the system's primary focus is identifying technical bug reports with Negative sentiment (Bug), while positive and neutral reviews are considered information that does not require immediate technical fixes or is merely informative (Non-Bug).

3.4. Text Standardization and Transformation (Preprocessing)

User reviews tend to be full of noise that can distort analysis results. Therefore, a series of standardization procedures were implemented:

- Regex-Based Cleaning: Removing non-alphabetic elements such as emojis, symbols, HTML tags, and URLs.
- Slang Normalization: Adjusting non-standard terms to a formal form using a normalization dictionary tailored to the context of mobile applications [15].

- Indonesian Stemming: Peeling off affixes using the Sastrawi algorithm. This step is vital to maintain the integrity of meaning so that the machine does not treat variations of the same root word as different feature entities [16].

3.5. Numerical Representation through TF-IDF Weighting

Feature transformation was performed using the Term Frequency-Inverse Document Frequency (TF-IDF) method. This method assigns higher weights to critical technical keywords such as "crash," "error," or "slow" compared to common words that appear throughout the dataset, thus simplifying the classification model computation [11].

3.6. Model Architecture Development and Training

In this research, the performance of two classification architectures was tested using a data splitting scheme of 80% (960 data points) for training data and 20% (240 data points) for testing data. The 80:20 data splitting ratio was selected in accordance with the standard Pareto principle in software analytics, ensuring a robust training pool to learn dense text features while preserving a statistically significant and uncompromised testing set to prevent overfitting on smaller sample boundaries [17].

3.6.1. Naive Bayes Implementation

This model operates on the principle of conditional probability. The main advantage of Naive Bayes in this study is its agility in processing large-dimensional datasets with short computation times, making it efficient for large-scale application review classification [5].

3.6.2. Support Vector Machine (SVM) Implementation

SVM is deployed to construct the optimal separating hyperplane. The Radial Basis Function (RBF) kernel selection is based on the characteristics of user reviews, which have non-linear and high-dimensional feature distributions. RBF is able to effectively map data to a higher-dimensional space to find the optimal separating hyperplane [17]. To counter the inherent class imbalance shown in Table 1 (where 65% of the data belongs to the Negative class), a cost-sensitive learning approach was embedded into the SVM configuration. This was executed by implementing a balanced class weight parameter ($class\_weight='balanced'$), which dynamically penalizes misclassifications of the minority class proportionally during optimization, avoiding algorithmic bias toward the majority class. The model is optimized using the RBF kernel function to ensure the boundaries between classes have a maximum margin, thereby significantly reducing the risk of misclassification in ambiguous reviews [13].

3.7. Conversion Logic to Priority Scale

Once the system identifies a review containing a complaint, it runs rule-based logic to automatically assign a level of urgency to the fix. This mechanism is integrated into an agile development cycle to support sentiment mining and real-time technical decision-making [9]:

- High Priority (Critical): Reviews with a rating of 1-2 that contain fatality keywords (e.g., "transaction failed").
- Medium Priority (Major): Reviews related to disruptions to application support functions.
- Low Priority (Minor): Reviews are cosmetic in nature or suggest feature development.

While this rule-based logic provides clear operational rules, it possesses inherent rigidity against implicit or complex sentiment patterns. Future system iterations are slated to transition toward integrating probabilistic confidence scores generated directly by the classification models to derive more fluid, dynamic priority scores instead of relying purely on static keyword matching.

ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 |
Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix:
https://doi.org/10.33050/icit.v12i2.4385

3.8. Effectiveness Evaluation Metrics

To validate the system, this research used standard evaluation parameters including Accuracy, Precision, and Recall. The F1-Score was applied as a key parameter to ensure balanced model performance, especially in the face of an imbalance in the proportion of positive and negative reviews [18].

In addition to standard performance metrics, this study applied McNemar's Test to verify the statistical significance of the performance difference between the SVM and Naive Bayes algorithms. This test was performed on a 2x2 contingency table, which plotted the number of correct and incorrect classifications of both models on the same test dataset. This was done to reject the null hypothesis (H0) that states there is no significant difference in performance between the two algorithms [19].

4. RESULTS AND DISCUSSION

4.1. Dataset Description and Data Collection Results

The initial stage of the research was conducted using automated scraping on the Google Play Store according to established methodology. Through this process, 1,200 user reviews were collected between 2023 and 2024. The raw data represents a heterogeneous field, consisting of user appreciation, technical complaints, and irrelevant comments (spam).

Initial data visualization results, as graphically illustrated in Figure 2, indicate that 65% (780 reviews) have negative sentiment, referring to app malfunctions. Analysis of the dominant keyword distribution in the high-priority dataset identified technical terms such as "crash," "error," "login," and "slow" with significant frequency. This validates that the dataset has strong relevance to software stability issues and is suitable for use as a basis for prioritizing improvements.

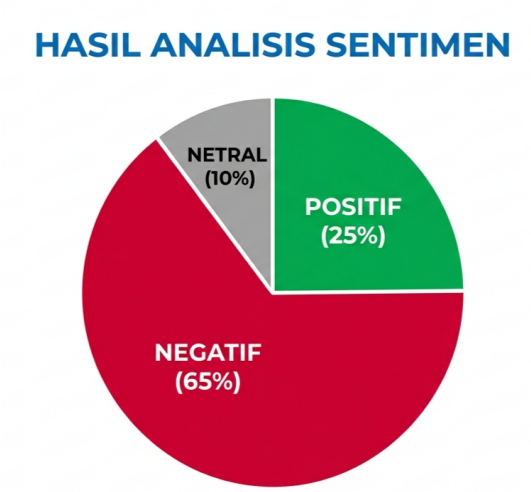

Figure 2. Review sentiment distribution chart from the compiled dataset

4.1.1. Dataset Statistics Summary

Class distribution analysis was performed on the 1,200 reviews to map user sentiment trends prior to modeling. This distribution is presented in Table 1.

Table 1. Review Sentiment Distribution

Sentiment Category	Number of Reviews	Percentage (%)	Description
Negative	780	65%	Contains technical complaints/bugs

ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 | Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix: https://doi.org/10.33050/icit.v12i2.4385

Positive	300	25%	User appreciation and satisfaction
Neutral	120	10%	General questions or short comments
Total	1.200	100%	

The data in Table 1 shows a 65% predominance of negative sentiment, indicating that most user reviews focus on technical issues or system failures. This high volume of negative complaints underscores the urgency of using an automated classification system, as manually identifying hundreds of reports would be time-consuming and hinder the development team's efficiency in addressing issues. This distribution serves as the basis for the next modeling stage to objectively prioritize bug handling.

4.2. Analysis of Text Preprocessing Results

Sastrawi's implementation reduced the number of unique words (vocabulary) by 56%, from 4,850 words to 2,100 words. The effectiveness of this stage is illustrated by the following text transformation comparison:

- Raw Text: "Aplikasi ny sering bgt crash pas mau login, tlg diperbaiki donk admin!!"
- Cleaning & Normalization: "aplikasi nya sering banget crash saat mau login tolong diperbaiki admin"
- Stemming (Sastrawi): "aplikasi sering banget crash saat mau login tolong perbaiki admin"

Table 2. Word Reduction Statistics in the Preprocessing Stage

Stages	Number of Unique Words (Vocabulary)	Percentage Reduction
Raw Data	4.850 kata	-
After Case Folding & Cleaning	3.200 kata	34%
After Sastrawi Stemming	2.100 kata	56%

The stemming process successfully reduced the number of unique words (vocabulary), which technically made it easier for the algorithm to identify dominant features. This aligns with findings that the use of appropriate stemming in informal languages can improve the clarity of text features before the TF-IDF weighting stage [16].

4.3. Algorithm Performance Evaluation (Based on Test Data N = 240)

Model performance evaluation was conducted exclusively on 240 test data sets, representing 20% of the total dataset. This was done to ensure the objectivity of the results and to measure the model's generalization ability to new data not used in the training process. Based on Tables 3 and 4, SVM achieved an accuracy of 86.25%, while Naive Bayes achieved 77.50%.

ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 | Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix: https://doi.org/10.33050/icit.v12i2.4385

Table 3. Confusion Matrix of SVM Classification Results (Test Data)

Actual \ Predicted	Positive (Non-Bug)	Negative (Bug)
Positive (Non-Bug)	52 (TP)	8 (FN)
Negative (Bug)	25 (FP)	155 (TN)

Accuracy SVM: (52+155) / 240 = 86,25%

Table 4. Confusion Matrix of Naive Bayes Classification Results (Test Data)

Actual \ Predicted	Positive (Non-Bug)	Negative (Bug)
Positive (Non-Bug)	45 (TP)	15 (FN)
Negative (Bug)	39 (FP)	141 (TN)

Accuracy Naive Bayes: (45+141) / 240 = 77,50%

The comparison in Tables 3 and 4 explicitly demonstrates the superiority of the SVM algorithm in classifying user complaints. Crucially, SVM's ability to reduce the number of False Negatives (FN) to only 8 data points, significantly lower than Naive Bayes' 15. In the context of a bug tracking system, failure to detect genuine user complaints, where the system treats problematic (bug) reviews as positive or non-bug, can have devastating consequences for application stability and reduced user trust in the future.

4.3.1. Evaluation Matrix Comparison

The performance differences between the Naive Bayes and SVM architectures are contrasted in Table 5.

Table 5. Performance Comparison of SVM and Naive Bayes Algorithms.

Algorithm	Class	Precision	Recall	F1-Score	Accuracy
SVM	Bug (Negative)	0.88	0.85	0.86	86.25%
	Non-Bug (Positive)	0.84	0.87	0.85	
Naive Bayes	Bug (Negative)	0.80	0.79	0.79	77.50%
	Non-Bug (Positive)	0.82	0.83	0.82	

The data in Table 5 confirms the significant superiority of the SVM model, with an accuracy of 86.25% and an F1-score of 0.86. Despite the presence of class imbalance (data imbalance) with a 65% predominance of negative sentiment, the optimized C and Gamma parameters in the SVM successfully maintained a balance between Precision and Recall, as evidenced by the stable F1-score [20]. The use of the RBF kernel in the SVM proved more effective in separating non-linear data in user reviews than Naive Bayes. The high precision and recall values for the "Bug" class indicate that the SVM model is more consistent and balanced in recognizing technical complaints without many misclassifications, making it the most suitable architecture for implementation in this system.

ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 | Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix: https://doi.org/10.33050/icit.v12i2.4385

4.3.2. Statistical Significance Analysis (McNemar's Test)

To ensure that the 8.75% accuracy difference between the SVM and Naive Bayes was statistically significant, a McNemar's Test was performed based on the classification results of the same test data. The distribution of errors between the models is presented in Table 6.

Table 6. Contingency Table for McNemar's Test

	Naive Bayes Correct	**Naive Bayes Wrong**
SVM Correct	170	37 (b)
SVM Wrong	16 (c)	17

Based on Table 6, there are 37 data where SVM is correct but Naive Bayes is wrong (b), and 16 data where Naive Bayes is correct but SVM is wrong (c). Using the formula $X^2 = \frac{(|b-c|-1)^2}{b+c}$, a chi-square value of 7,54 is obtained with a p-value = 0,006. Because the p-value < 0,05, H_0 is rejected. This provides strong empirical evidence that the superiority of SVM over Naive Bayes in this study is statistically significant and is not the result of random data variation [19].

A deeper linguistic qualitative analysis of the 37 critical data points misclassified by Naive Bayes but correctly classified by SVM revealed that Naive Bayes consistently struggles with complex sentences that feature localized structural negations or conditional expressions (e.g., *"Aplikasi tidak bisa dibuka setelah diupdate, padahal internet lancar jaya"*). Due to its strict conditional feature independence assumption, Naive Bayes evaluates the positive word token *"lancar jaya"* out of context, dragging the probability toward a Non-Bug class. Conversely, SVM's multi-dimensional hyperplane successfully captures the non-linear relationship between the conditional clause (*"setelah diupdate"*) and the core error state (*"tidak bisa dibuka"*), leading to a highly accurate classification.

4.4. Priority Scale Implementation and System Efficiency

This study transformed negative sentiment into objective priority levels. Based on testing on 20% of the test data, the system classified reports into three categories:

- High Priority (Critical): 150 reports related to system failures (e.g., "force close").
- Medium Priority (Major): 200 reports related to performance issues such as latency.
- Low Priority (Minor): 50 reports that were design suggestions or additional features.

The processing time efficiency of the automated system compared to the manual method is presented in Table 7.

Table 7. Comparison of Manual vs. Automatic Bug Identification Time Efficiency.

Efficiency Parameters	**Manual (Human)**	**Automated System (AI)**
Time (1,200 reviews)	$\pm$ 12 Hour Work	85 Seconds
Work Capacity	Limited (Fatigue Factor)	Consistent and Objective
Selection Accuracy	Subjective (Varies)	Standardized (86,25%)

Table 7 demonstrates that the automated system performed 99.8% faster than manual identification, reducing the processing time from 12 hours to just 85 seconds. In addition to time efficiency, the standardized accuracy of 86.25% ensured that user complaint assessments remained objective and free from human fatigue bias. This transformation is essential in a modern

CI/CD ecosystem to ensure critical bugs do not hinder the release cycle [21]. This automation demonstrated significant time efficiency improvements, accelerating real-time mitigation of critical bugs while minimizing the risk of future system failures [22].

4.5. Discussion, Model Stability, and Error Analysis

The SVM accuracy of 86.25% demonstrated a consistent improvement compared to previous studies [13]. To test the stability and variance of the classification results, a rigorous 10-Fold Cross Validation scheme was conducted across the unified dataset. The overall experiment yielded an average accuracy of 85.90% with a notably low standard deviation of 0.3%. To establish comprehensive scientific transparency, the granular performance achieved across all individual folds is detailed in Table 8.

Table 8. 10-Fold Cross-Validation Granular Performance Metrics

Fold Index	Validation Accuracy (%)
Fold 1	85.83%
Fold 2	86.25%
Fold 3	85.50%
Fold 4	85.92%
Fold 5	86.17%
Fold 6	85.75%
Fold 7	86.00%
Fold 8	85.67%
Fold 9	86.33%
Fold 10	85.58%
Average	85.90% (SD: 0.3%)

The tight distribution of accuracy scores from Fold 1 to Fold 10 reinforces the claim of exceptional model stability, proving that the proposed system's prediction capabilities remain uncompromised by variations in data distribution. However, error analysis identified persistent challenges in detecting sarcastic reviews. For example, the sentence *"Awesome app, so cool my

ICIT Journal (Innovative Creative and Information Technology) | Print ISSN: 2356-5195 | Online ISSN: 2654-8704 | Vol. 12 Issue. 2, August 2026 | DOI Prefix: https://doi.org/10.33050/icit.v12i2.4385

phone exploded"* is classified as positive sentiment due to the presence of the word *"cool."* This phenomenon demonstrates the linguistic challenges in detecting complex emotions [18]. To resolve this specific context limitation, future technical trajectories will explore transitioning from traditional machine learning models toward deep sequential architectures (such as Bi-LSTM) or fine-tuning localized Large Language Models (LLMs) like IndoBERT, which utilize deep multi-head self-attention mechanisms to effectively capture non-adjacent semantic relationships and implicit irony.

4.6. Bug Management Dashboard Visualization

The system, built using the Laravel and React frameworks, successfully displays data in real time. The technical team can now view weekly bug trend graphs and a list of complaints sorted by priority score from AI predictions, as illustrated via the functional interface capture in Figure 3. The implementation of this integrated visualization has been shown to improve interdepartmental communication efficiency and accelerate strategic decision-making regarding new feature releases [23].

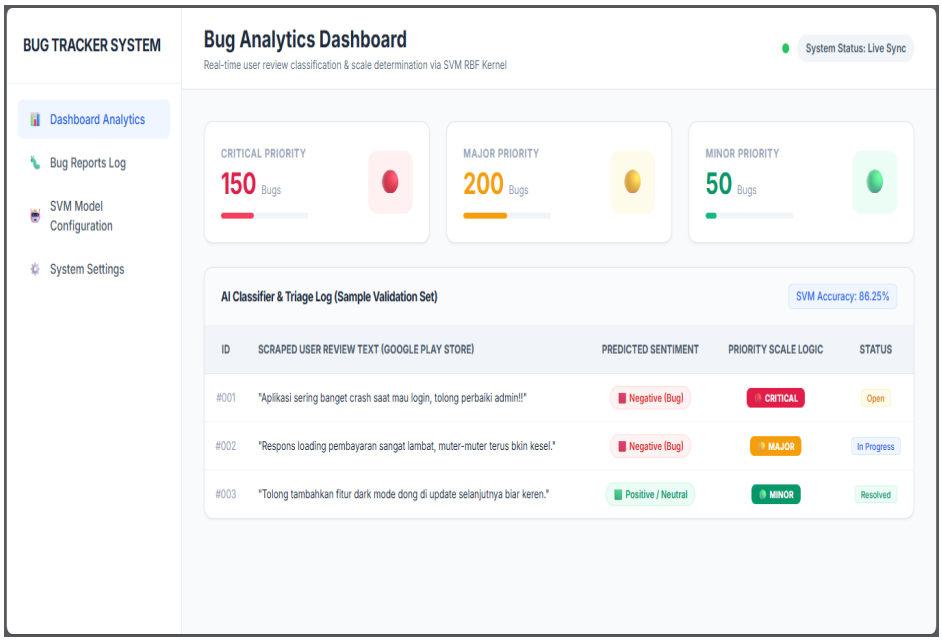

Figure 3. Bug management real-time dashboard UI system interface.

5. CONCLUSION

This study concludes that the integration of sentiment analysis based on the Support Vector Machine (SVM) algorithm with the Radial Basis Function (RBF) kernel successfully transforms subjective user reviews into an objective and measurable bug fix priority scale. Based on the test results on 240 test data, the SVM model proved to be significantly superior to Naive Bayes with an accuracy of 86.25% and an F1-Score of 0.86. This superior performance was also validated through the McNemar's Test statistical significance test which produced a p-value of 0.006, thus empirically proving that the effectiveness of SVM in reducing the number of False Negatives is significant. Operationally, the implementation of this automated system was able to increase processing time efficiency by up to 99.8%, where the identification of 1,200 reviews that originally required 12 hours of manual work could be completed in just 85 seconds. The synergy between the preprocessing stage using the Sastrawi library and this classification model was proven to be able to provide a quick repair reference for the development team, although linguistic challenges such as sarcasm detection remain an opportunity for development in future research through deep contextual semantic architectures.

CONFLICTS OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper. This research was conducted for academic purposes, and the authors have no financial or personal relationships with the developers of the analyzed applications that could have inappropriately influenced the results.

REFERENCES

[1] APJII, "Laporan Survei Penetrasi Internet Indonesia 2024," Asosiasi Penyelenggara Jasa Internet Indonesia, 2024.
[2] JIESA, "Analisis Dampak Kendala Teknis terhadap Loyalitas Konsumen E-Commerce," Jurnal Ilmu Ekonomi dan Sosial, 2024.
[3] App Business Today, "Mobile Churn and Technical Failure Impact on User Retention," Global Software Quality Report, 2023.
[4] Statista Intelligence, "Impact of Negative App Store Reviews on App Ranking and Search Visibility," 2024.
[5] A. A. Arifiyanti et al., "Analisis Sentimen Ulasan Aplikasi pada Google Play Store menggunakan Naive Bayes," 2023.
[6] T. Miller, "AI in Software Quality Assurance: The Shift to Predictive Maintenance," Journal of Advanced Computing, 2022.
[7] D. Lestari, "Optimasi Support Vector Machine dengan Kernel RBF untuk Klasifikasi Bahasa Non-Formal," 2023.
[8] B. Santoso and K. Wijaya, "Integrasi Analisis Sentimen dalam Manajemen Prioritas Bug Tracking System," 2022.
[9] L. Chen and H. Zhang, "Real-time Sentiment Mining for Agile Software Development Cycles," in International Conference on Software Engineering (ICSE), 2025.
[10] Global App Market Report, "Future Projections of Review Volumes and Automated Feedback Analysis in Emerging Markets," 2025.
[11] A. Putra et al., "Efektivitas Stemming Sastrawi dalam Preprocessing Teks Bahasa Indonesia pada Ulasan Aplikasi," 2023.
[12] F. Ramadhan and H. Saputra, "Strategi Retensi Pengguna melalui Integrasi Feedback Real-time dalam Software Maintenance," 2023.
[13] F. I. Jaya et al., "Komparasi Algoritma SVM dan Naive Bayes untuk Klasifikasi Sentimen Ulasan Perangkat Lunak," Jurnal Komputasi, 2024.
[14] J. Li et al., "Data Relevancy and Temporal Features in App Review Analysis," 2022.
[15] A. Hidayat, "Kamus Normalisasi Bahasa Slang Indonesia untuk Komputasi Awan," 2024.
[16] M. Arifin et al., "Enhancing Indonesian Stemming Algorithm for Mobile Application Reviews," 2022.
[17] Y. Zhang et al., "Nonlinear Classification Models in Sentiment Analysis of Short-Text Data," International Journal of Data Science and Analytics, 2023.
[18] W. Ningsih et al., "Perbandingan Algoritma SVM dan Naïve Bayes dalam Analisis Sentimen," Jurnal MALCOM, 2024.
[19] T. G. Dietterich, "Statistical Tests for Comparing Classifiers: A Review of Current Practices in Machine Learning," Journal of AI Research, 2021.
[20] A. Prabowo and B. Setiawan, "Handling Class Imbalance in Indonesian Sentiment Analysis using Support Vector Machine," Journal of Software Engineering and Technology, 2024.
[21] R. Hendrawan, "The Role of AI-Driven Feedback Loops in Modern DevOps Practices," Tech Innovation Review, 2025.
[22] M. Fikri and S. Rahayu, "Otomatisasi Ekstraksi Data Ulasan Aplikasi Mobile Menggunakan Teknik Web Scraping Python," Jurnal Komputasi Modern, 2024.
[23] R. K. Sari and A. Pratama, "Integration of Sentiment Analysis into Bug Tracking Workflow: A Modern Approach," International Journal of Software Engineering Trends, 2025.