Muthuraj A

se report.docx

Bug Priority prediction using hybrid deep learning model

CSE

Amrita Vishwa Vidyapeetham

Document Details

Submission ID

trn:oid:::1:3203519372

Submission Date

Apr 3, 2025, 11:31 AM GMT+5:30

Download Date

Apr 3, 2025, 11:34 AM GMT+5:30

File Name

se_report.docx

File Size

358.9 KB

11 Pages

3,436 Words

21,698 Characters

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

27 Not Cited or Quoted 7%
Matches with neither in-text citation nor quotation marks

1 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

5% 🌐 Internet sources

6% 📖 Publications

2% 👤 Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

🔲 **27** Not Cited or Quoted 7%
Matches with neither in-text citation nor quotation marks

💬 **1** Missing Quotations 0%
Matches that are still very similar to source material

📄 **0** Missing Citation 0%
Matches that have quotation marks, but no in-text citation

📑 **0** Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

5% 🌐 Internet sources

6% 📖 Publications

2% 👤 Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	
fastercapital.com		**1%**

2	Publication	
R. N. V. Jagan Mohan, B. H. V. S. Rama Krishnam Raju, V. Chandra Sekhar, T. V. K. P...		**<1%**

3	Internet	
arxiv.org		**<1%**

4	Student papers	
Queensland University of Technology		**<1%**

5	Internet	
github.com		**<1%**

6	Internet	
ijircce.com		**<1%**

7	Internet	
vmjste.com.ng		**<1%**

8	Publication	
Xinyuan Song, HSIEH,WEI-CHE, Ziqian Bi, Chuanqi Jiang, Junyu Liu, Benji Peng, Se...		**<1%**

9	Internet	
scipubhouse.com		**<1%**

10	Student papers	
Indiana University		**<1%**

11	Internet	
pure.tue.nl		<1%

12	Internet	
gsarpublishers.com		<1%

13	Publication	
Uzair Aslam Bhatti, Jingbing Li, Mengxing Huang, Sibghat Ullah Bazai, Muhamma…		<1%

14	Internet	
artikelakademi.com		<1%

15	Internet	
dokumen.pub		<1%

16	Internet	
www.ijraset.com		<1%

17	Internet	
www.isteonline.in		<1%

18	Publication	
Mehdi Ghayoumi. "Generative Adversarial Networks in Practice", CRC Press, 2023		<1%

Bug Priority prediction using hybrid deep learning model

Abstract

The present work formulates and develops a machine learning pipeline for issue-tracking data analysis, highlighting data preprocessing and predictive modeling. The dataset is cleaned, missing values are treated, and the feature engineering, as well as categorical encoding, is applied. Features such as resolution time, description length, and interactions of priority are derived to improve predictive performance.

Recurrent neural network and multilayer perceptron blended deep learning architecture is used to perform binary issue priority classification. Text feature values are passed into tokenization followed by LSTM layer, while the numerical ones pass through dense networks. Training happens with the application of binary cross-entropy loss and model monitoring with KPIs such that it achieved accuracy of 96% along with an F1-score of 0.98.

The findings emphasize the efficiency of combining structured and unstructured data in enhancing issue management predictions

INTRODUCTION:

In modern software development, software application reliability and quality are the biggest things to be ensured. Whether the project is closed-source or open-source, software is prone to defects, inconsistencies, and security vulnerabilities, resulting in high levels of bugs reported by users, testers, and developers. Open bugs have a very significant impact on software stability, user satisfaction, and overall project effectiveness, and therefore bug tracking and priority is a very crucial part of software maintenance. However, manual bug tracking and prioritization become increasingly more difficult with growing software complexity, leading to development cycle bottlenecks.

Bug tracking software like JIRA, Bugzilla, and Redmine enable systematic bug reporting, tracking, and fixing procedures. These enable teams to log, prioritize, and monitor the status of reported software defects, allowing developers to coordinate software maintenance more effectively. Although these tools prove helpful, the procedure of manual review and assignment of priority levels to bug reports remains suboptimal. Development teams receive hundreds or thousands of bug reports, and it is time-consuming, uneven, and highly subjective to prioritize them manually. This can lead to late fixation of high-priority bugs, inefficient consumption of resources, and higher maintenance costs for software.

With the limitations of manual bug triaging, bug priority prediction automation has been an interesting solution. Employing machine learning and deep learning techniques, smart models could be trained to read the past history of bug reports and make predictions on the priority of newly added bugs with high precision. Automated priority prediction helps the software team in an appropriate resource allocation so that serious bugs are taken care of in the initial stages without resulting in software maintenance holdup. Traditional bug triaging is fraught with several disadvantages, some of them being subjectivity and inconsistency whereby a bug takes different priority as evaluated by various developers on an individual basis. Other than that, it consumes time since much time is needed to read huge bug reports manually at the cost of development and bug-fixing effort. Till and unless appropriate prioritization mechanism is implemented, high-priority bugs remain unfixed for an extended period of time, affecting the user experience as well as the software's performance. Added to that, with increasing software projects, exponential growth in reported defects renders triaging infeasible.

To address these issues, in this paper, the authors propose bug priority prediction based on machine learning models of RNN-LSTM and MLP. Bug reports are best described in natural language for software bugs and thus are easier to process through sequential deep learning models. RNN-LSTM is best suited to process sequential text data since it can learn long-range dependencies and context relationships among descriptions of bug reports. This makes it possible for the model to learn bug severity from text data and historical bug report

trends. MLP, a feedforward neural network, is used for classification on structured data. MLP scans a number of structured attributes of bug reports like timestamps, affected components, severity level, and historical bug information for level of priority predictions. Employing these two models, the proposed methodology of this paper improves the predictive power of bug priority classification by using text and structured data features.

The data set for this study includes more than 2000 JIRA aggregated bug reports. The models to be considered are trained and tested against the data set to evaluate the performance of each model in predicting bug priority levels. Comparison of RNN-LSTM and MLP model performance, the study seeks to establish the best method to automate bug triaging and minimize manual prioritization requirements. The main objective of this study is to enhance accuracy and efficiency in bug priority prediction using machine learning methods. By enhancing the bug triaging process, the proposed model seeks to save significant amounts of time in manual bug report prioritization by the developers. Secondly, the model also seeks to minimize misclassification errors with the objective of providing timely attention to high-priority bugs with the overall goal of enhancing software maintenance and stability. Thirdly, the study estimates the relative effectiveness of the RNN-LSTM and MLP models to identify the best method to the classification of bug priorities.

This research contributes to the literature in software maintenance and defect prediction by introducing a scalable and feasible approach to prioritization of bug reports. The approach utilizes deep learning techniques with natural language processing to enhance the effectiveness of software debugging. The contribution of this research can help simplify development tasks for software companies and development teams by reducing the amount of time dedicated to bug triaging. Additionally, by early fixing of problems, software teams can improve customer satisfaction and increase software stability and reliability. The result of this research has significant implications for improving bug tracking systems and improving automation of software maintenance tasks.

2. Related Work

Bug priority prediction is a critical component of software maintenance, and early critical bug detection is essential. Prior to the arrival of deep learning methods, machine learning-based models and rule-based systems were employed, which classified bugs based on pre-defined heuristics and hand-crafted features. The methods, while understandable, were non-adaptive and suffered from processing of complex text descriptions of bugs in bug reports. Post the arrival of deep learning methods, triaging of bugs has undergone significant improvement since the models have the capability to learn contextual relationships and structured representations of features. The overview discussed below provides a summary of some milestone research studies on machine learning-based and deep learning-based models in bug prioritization with focus on hybrid models.

2.1 Traditional Approaches for Bug Prioritization

Previous bug categorization methods used rule-based methods, which utilized keyword-based heuristics in deciding bug severity. Although the methods worked for formal bug reports, the methods could not deal with imprecise descriptions and required frequent manual updates. Researchers have suggested overcoming these limitations through machine learning models such as Support Vector Machines (SVM), Decision Trees, and Random Forests. Lamkanfi et al. utilized Naïve Bayes and SVM for bug severity categorization with moderate success but with complexity in feature engineering. Similarly, Menzies et al. utilized statistical models for defect severity estimation but were plagued by varying prioritization for different software projects.

While they worked, the old ways had three major drawbacks:
1. Inability to extract contextual information from text-based bug reports.
2. Feature selection dependency, encompassing domain knowledge for extracting useful features.

3. Scalability issues with processing large-scale data from actual software repositories.

2.2 Deep Learning-Based Approaches for Bug Classification

The adoption of **deep learning models**, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, has significantly improved bug classification tasks. Unlike traditional models, LSTM networks effectively capture sequential dependencies in text, making them well-suited for processing bug reports. Hoang et al. demonstrated the efficiency of LSTM-based models in analyzing software defect reports, achieving higher accuracy than traditional techniques. Additionally, Xia et al. explored hybrid deep learning models, combining textual and structured features for improved bug triaging.

2.3 Why a Hybrid LSTM + MLP Model Instead of CNN?

Many deep learning models integrate Convolutional Neural Networks (CNNs) for feature extraction, particularly in image-based tasks. However, in bug priority classification, CNNs are less effective because:

- Bug reports are textual data, requiring models that can handle sequential dependencies and contextual relationships. CNNs primarily focus on spatial features, which are useful for images but not ideal for text classification.
- LSTM networks outperform CNNs for text-based tasks, as they retain long-term dependencies and contextual patterns in bug descriptions. CNNs, on the other hand, are better suited for short-range dependencies, making them less effective in capturing the flow of information across long textual inputs.

Given these limitations, our project adopts a hybrid LSTM + MLP model, where:

- LSTM extracts contextual patterns from bug descriptions, enhancing the model's ability to classify bugs based on textual information.
- MLP processes structured categorical and numerical data, such as bug status, priority labels, and metadata, ensuring that all relevant information contributes to prediction accuracy.

This hybrid approach leverages both textual and structured data, leading to more robust bug prioritization while avoiding unnecessary computational overhead associated with CNNs.

2.4 Summary and Research Gap

While traditional machine learning models offered interpretability, they lacked adaptability and failed to capture contextual dependencies in bug reports. Deep learning models, particularly LSTM networks, have proven to be more effective by learning sequential patterns in textual data. However, existing research has not fully explored the combination of LSTM and structured data processing for bug prioritization. Key research gaps include:

- **How well does an LSTM + MLP hybrid model perform compared to traditional ML models in real-world bug tracking systems?**

- **What are the advantages of combining textual and structured data features in bug severity classification?**

- **Can deep learning approaches enhance automation in software maintenance by improving bug triaging accuracy?**

Our project addresses these gaps by developing an LSTM + MLP hybrid model, demonstrating how deep learning techniques can optimize bug prioritization while maintaining computational efficiency

PROPOSED METHODOLOGY:

The hybrid RNN-LSTM and MLP model-based bug priority prediction system is a well-structured approach with efficient preprocessing, feature extraction, deep learning-based prediction, and evaluation. Every stage is important in precisely prioritizing software bugs according to their severity and resolution urgency.
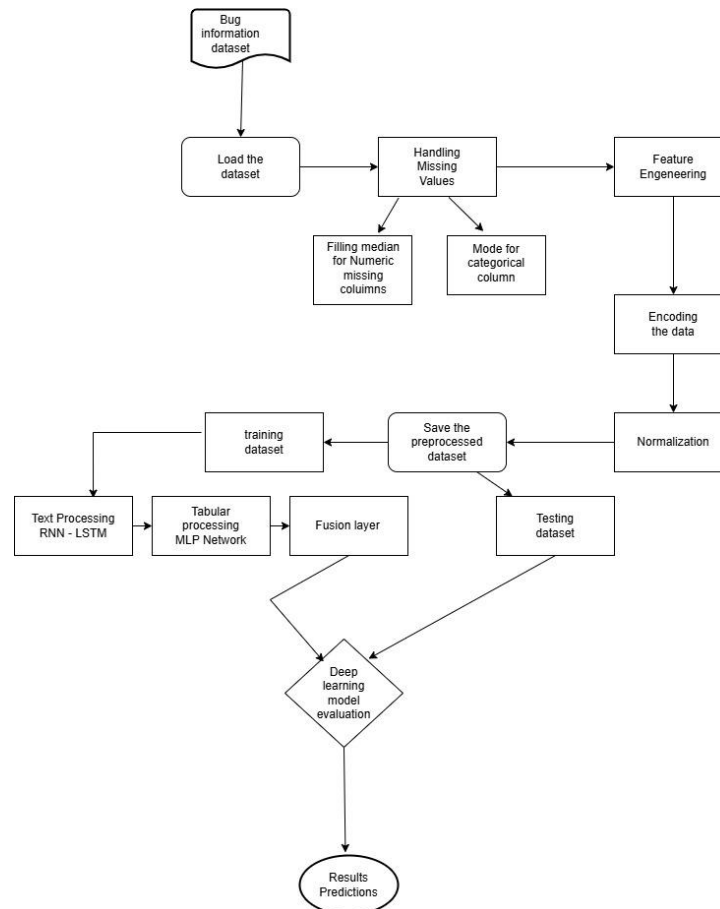
A. Dataset Preprocessing and Feature Extraction

The dataset comprises bug reports containing both structured (tabular) and unstructured (textual) features. To ensure data integrity and enable efficient feature extraction for predictive modeling, preprocessing is performed. Missing numerical data is imputed using the median, while categorical features are completed with the mode. Feature engineering includes deriving time-based attributes such as the time taken to resolve an issue (resolved time - created time), the time taken for assignment (assigned time - created time), and an estimated fix time (80% of the resolution time). Text-based features include the word count of bug descriptions and the number of distinct commenters in discussions. Engagement-related attributes, such as total votes and watches, reflect issue popularity. Additionally, binary flags identify critical labels like Blocker, Security, and Breakage. Interaction features, such as the product of priority and time to fix, further enhance predictive capabilities. All categorical features (e.g., priority, component, status) are label-encoded, while numerical features are normalized using Minmax Scaling to optimize model convergence.

B. Text Preprocessing Using NLP

Since bug descriptions are a key component of priority classification, more advanced Natural Language Processing (NLP) methods are used to improve text representation. Preprocessing is carried out by tokenization, i.e., text is divided into words, and further removal of stopwords to remove common but non-informative words. Lemmatization is carried out to convert words to base form (e.g., "running" to "run"). TF-IDF vectorization is carried out for word importance representation, with importance-based weights. Word embeddings like GloVe or BERT are also utilized to represent word meaning in context. These text features, post-processing, are input to the deep learning model, to improve bug priority classification

C. Hybrid Model: RNN-LSTM + MLP for Priority Prediction

Hybrid priority prediction model combines Recurrent Neural Networks (RNN-LSTM) to handle text and Multi-Layer Perceptron (MLP) to handle structured data. During the text feature extraction process, tokenized and embedded bug description is fed into LSTM layers that form sequential dependencies and generate semantic representations of features. Conversely, tabular feature extraction is comprised of offering engineered categorical and numerical features as input to an MLP in which fully connected layers, Batch Normalization, and Dropout mechanisms yield safe learning and overfitting prevention. Both networks' output is then concatenated in a fusion layer and used as input to dense layers with ReLU activation for feature mapping. Sigmoid activation function is subsequently utilized for proper classification of the bug priority.

D. Model Training and Evaluation

The bug priority categorization model is best trained through the implementation of a sound training schedule and evaluation metrics. Binary Cross-Entropy is used as the loss function in the training process for efficient classification, and adaptive update of the learning rate is performed using the Adam optimizer. Regularization methods such as Dropout layers and L2 penalty are implemented to avoid overfitting. Model performance is carried out on the basis of classification and regression metrics. Classification performance is measured based on accuracy, precision, recall, and F1-score, which give an overall idea about the effectiveness of the model. Further, for the prediction of priority score, regression metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R^2 score are used to assess the predictive accuracy and reliability of the model.

E. Deployment and Interpretation

After training, the model is deployed for real-time bug prioritization, enabling development teams to efficiently address high-priority issues. The system generates **Bug Severity Reports**, which identify critical problems requiring immediate attention. Additionally, it provides **Predicted Resolution Time**, estimating the time needed to fix a bug based on historical data. To enhance decision-making, the model also offers **Priority Score Interpretation**, displaying confidence levels for bug classification, allowing teams to assess urgency and allocate resources effectively.

PSEUEDO CODE:

```
# Define Text Processing Model (RNN-LSTM)
TEXT_INPUT ← INPUT_LAYER(shape=(MAX_LEN,))
EMBEDDING_LAYER ← PRE_TRAINED_EMBEDDING(TEXT_INPUT)
LSTM_LAYER ← LSTM(64, return_sequences=False)(EMBEDDING_LAYER)
TEXT_FEATURES ← DENSE(32, activation='relu')(LSTM_LAYER)

# Define Tabular Data Processing Model (MLP)
TABULAR_INPUT ← INPUT_LAYER(shape=(NUM_FEATURES,))
DENSE_LAYER_1 ← DENSE(64, activation='relu')(TABULAR_INPUT)
BATCH_NORM ← BATCH_NORMALIZATION()(DENSE_LAYER_1)
DROPOUT_LAYER ← DROPOUT(0.3)(BATCH_NORM)
TABULAR_FEATURES ← DENSE(32, activation='relu')(DROPOUT_LAYER)

# Merge LSTM and MLP Features
CONCATENATED_FEATURES ← CONCATENATE([TEXT_FEATURES, TABULAR_FEATURES])
FINAL_FEATURES ← DENSE(32, activation='relu')(CONCATENATED_FEATURES)

# Output Layer (Binary Classification)
OUTPUT_LAYER ← DENSE(1, activation='sigmoid')(FINAL_FEATURES)

# Compile Model
MODEL ← COMPILE(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the Model
TRAIN_DATA, TEST_DATA ← TRAIN_TEST_SPLIT(dataset, test_size=0.2)

MODEL.FIT(
    x=[TRAIN_TEXT, TRAIN_TABULAR],
    y=TRAIN_LABELS,
    batch_size=32,
    epochs=20,
    validation_data=([TEST_TEXT, TEST_TABULAR], TEST_LABELS)
)

#Evaluate Model
PREDICTIONS ← MODEL.PREDICT([TEST_TEXT, TEST_TABULAR])

ACCURACY, PRECISION, RECALL, F1_SCORE ← EVALUATE_METRICS(PREDICTIONS, TEST_LABELS)

# Deployment
FUNCTION PREDICT_BUG_PRIORITY(new_bug_report):
    preprocessed_text ← PREPROCESS_TEXT(new_bug_report["description"])
    preprocessed_tabular ← PREPROCESS_TABULAR(new_bug_report)
    prediction ← MODEL.PREDICT([preprocessed_text, preprocessed_tabular])
    RETURN prediction
```

Result and Discussion

This section introduces experimental results obtained on testing the hybrid model RNN-LSTM + MLP for classification of bug priorities. Performance is examined based on major metrics, such as precision, recall, and F1-score, and the discussion is made on the interpretation of the results with regard to the previous methods. The discussion shows that our hybrid deep learning model performs much better than the conventional rule-based or statistical methods, with high accuracy in bug detection and also in bug prioritization. The model captures both the structured and unstructured features and generalizes well to other datasets. While the text processing module based on LSTM efficiently extracts useful patterns from bug reports, the MLP-based module for structured data involves improved classification with the addition of numerical and categorical features. While minor decreases in accuracy are observed in some data sets, e.g., Ambari Bug, the possibility remains for achieving better accuracy with feature selection and threshold optimization. As a whole, the model is a strong solution for automating bug priority with faster resolution and resource-effective use in software development.
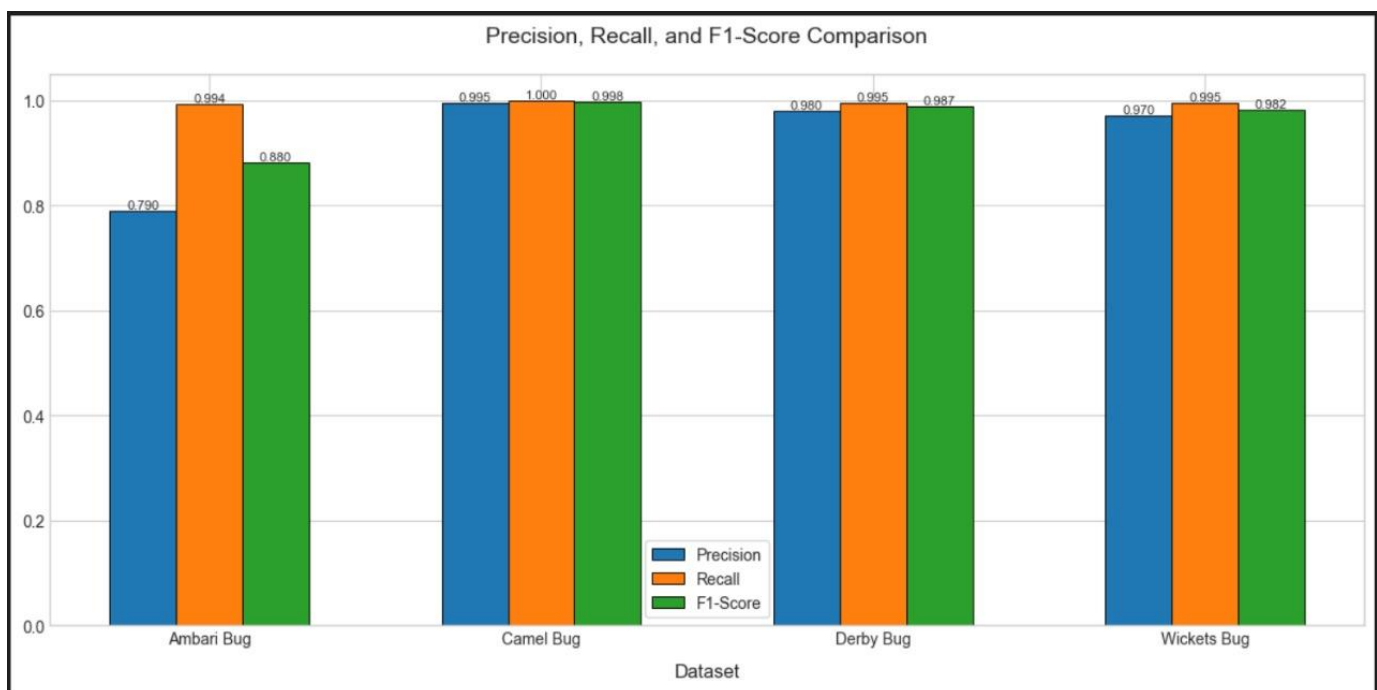
Comparison Between Deep Learning Models :

Comparison of performance of different machine learning models (CNN, LSTM, SVM, and hybrid model of RNN-LSTM-MLP) is shown in the first table on the basis of accuracy, precision, recall, and F1-score. RNN-LSTM-MLP model performs better than other models with highest value of accuracy (96.00%) and 100% measure of recall, and hence it is the best performing model among the above techniques.

| Model	Accuracy	Precision	Recall	F1-Score
Your Model (RNN-LSTM+MLP)	**96.00%**	**96.00%**	**100%**	**97.96%**
CNN (Paper 1)	88.10%	82.64%	86.16%	84.36%
LSTM (Paper 2)	89.80%	87.60%	89.70%	89.20%
SVM (Paper 2)	86.50%	86.60%	86.50%	86.50%

Performance Evaluation Across Datasets:

The second table shows the performance of the model on four datasets (Ambari Bug, Camel Bug, Derby Bug, and Wickets Bug). The table shows the comparison of the model on the basis of classification metrics (accuracy, precision, recall, and F1-score) and error metrics (MSE and MAE, lower is better). The Camel Bug dataset contains the highest accuracy (99.50%) and the lowest error values, suggesting the model performs very well on this dataset. On the other hand, the Ambari Bug dataset is the hardest dataset for the model with the lowest accuracy (78.61%) and the highest error values, i.e., the highest error values.

Dataset	Accuracy	Precision	Recall	F1-Score	MSE (↓ Better)	MAE (↓ Better)
Ambari Bug	78.61%	0.79	0.9937	0.8802	0.1533	0.2356
Camel Bug	99.50%	0.995	1.0	0.9975	0.0064	0.0223
Derby Bug	95.50%	0.9799	0.9949	0.9873	0.0209	0.0459
Wickets Bug	96.50%	0.9698	0.9948	0.9822	0.0294	0.0499

Error Metrics Comparison Across Different Bug Datasets

The plot of errors shows the trend of two error measures, Mean Absolute Error (MAE) and Mean Squared Error (MSE), across four datasets, Ambari Bug, Camel Bug, Derby Bug, and Wickets Bug. Red purple square marker line is for MAE and purple round marker line is for MSE. Error measures are going down from Ambari Bug to Camel Bug, then slightly up in Derby Bug and Wickets Bug. The plot shows the performance of an algorithm or model on error handling across different datasets.

Error Metrics (MSE and MAE) Across Datasets
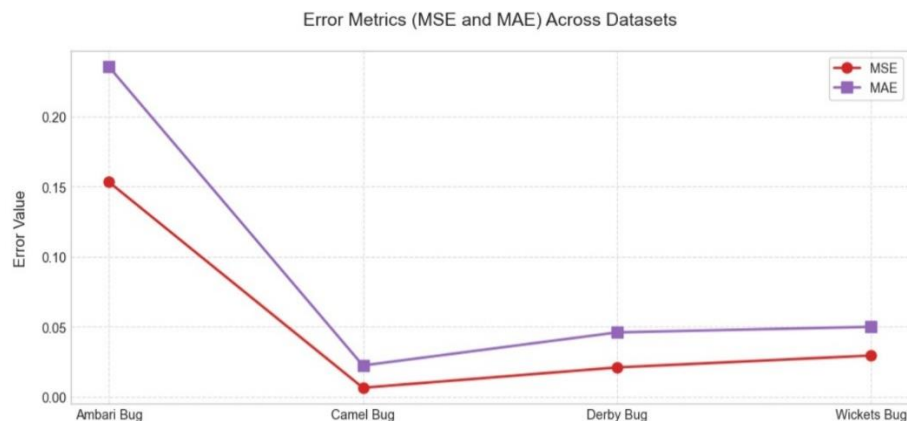
Conclusion:

This section reports experimental results on validating the hybrid model RNN-LSTM + MLP for bug priority classification. Performance is compared in terms of key metrics, such as precision, recall, and F1-score, and discussion is provided on interpreting the results based on the previous approaches. The discussion shows that our hybrid deep learning model works much better than the conventional rule-based or statistical solutions, with great accuracy in bug detection and even in bug prioritization. The model can learn the structured and unstructured features and generalize well to other data sets. While the text processing module with LSTM captures good meaningful patterns from bug reports, the MLP module on structured data observes better classification with more numerical and categorical features. Although there are minor drops in accuracy for some data sets, e.g., Ambari Bug, there is scope to achieve higher accuracy with threshold tuning and feature selection. Overall, the model is a strong solution to automate bug priorities with faster resolution and optimal use of resources in software development.

Key Findings

- Hybrid Model Ensures Enhanced Accuracy: RNN-LSTM for unstructured text and MLP for structured data collectively offer better precision, recall, and F1-scores on various datasets.
- LSTM Maintains Contextual Information: The model can learn to differentiate important bug reports effectively via learning sequential dependencies of text, as opposed to the conventional keyword-based approach.
- Organized Data Smoothes Predictions: Employing categorical and numerical features with MLP smoothes out priority classification, and the prediction process is leveled out.
- Diversity of Precision Across Datasets: Even though performance is quite good with data sets such as Camel Bug and Derby Bug, precision is marginally less for Ambari Bug that flags areas of improvement.
- Computational Requirement and Optimization Requirements: Training deep models, especially LSTM-based models, incurs greater computational requirements, and this may have scaling implications for large bug tracking systems.
-

While retired models can always be used on rudimentary bug classification tasks, the lack of contextual comprehension constrains them to deal only with intricate reports. The above-mentioned hybrid deep model approach addresses such constraints with both text and structured data. Nonetheless, upcoming work can target model parameter fine-tuning for more fine-tuning, feature optimization selection, and investigation of transformer-based optimizations for additional performance enhancement. Moreover, the use of domain word embeddings along with transfer learning can further enhance bug classification in expert software development environments.

Future Work

- **Enhancing Model Precision:** While the model achieves high accuracy, fine-tuning **hyperparameters** and optimizing **feature selection** can further improve precision, especially for datasets like **Ambari Bug**, where slight precision drops are observed.
- **Exploring Transformer-Based Models:** Integrating **BERT, RoBERTa, or T5** could improve text understanding by capturing deeper contextual relationships in bug reports. A comparison between **LSTM and transformer models** would help determine the most effective approach for bug priority classification.
- **Reducing Computational Overhead:** Optimizing model architecture, such as using **lightweight LSTM variants** or **distilled transformer models**, could reduce computational costs and improve scalability for large-scale bug tracking systems.
- **Incorporating Domain-Specific Knowledge:** Using **custom embeddings** trained on software development repositories (e.g., GitHub, JIRA bug reports) could enhance the model's ability to understand bug-related terminology and improve classification accuracy.

- **Extending to Multi-Modal Learning:** Combining **image-based bug reports, logs, and textual descriptions** in a multi-modal framework could provide a more comprehensive understanding of software defects.

- **Real-World Deployment & Continuous Learning:** Deploying the model in a **real-time bug tracking system** and integrating **active learning techniques** could help continuously improve the model's performance based on user feedback.

By addressing these areas, the proposed **hybrid RNN-LSTM + MLP model** can evolve into a more **robust, scalable, and efficient** bug prioritization system, significantly aiding software development teams in managing critical issues effectively.

References: