**Bug Priority prediction using hybrid deep learning model**

*A report submitted in partial fulfilment for the Assignment of*

*22AIE311 Software Engineering (Project Based)*

By

**A. CHARITH: AIE22001 D.PARTHAVA NAIDU: AIE22010 D.SIRDHAR ADITHYA: AIE22049**



Under the Supervision of

**Dr. Chandan Kumar**

Assistant Professor (Sl. Gr.)



School of Computing

Amrita Vishwa Vidyapeetham, Amaravati Campus

Andhra Pradesh, 522503, India

2025

**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, ensuring the reliability and quality of software applications is a critical challenge. Whether in open-source or closed-source projects, software is prone to defects, inconsistencies, and security vulnerabilities, leading to a large number of bug reports filed by users, testers, and developers. The presence of unresolved bugs significantly impacts software stability, user satisfaction, and overall project efficiency, making bug tracking and prioritization an essential aspect of software maintenance. However, manually managing and prioritizing bug reports becomes increasingly difficult as software complexity grows, leading to bottlenecks in development cycles.

Bug tracking systems such as JIRA, Bugzilla, and Redmine facilitate structured bug reporting, monitoring, and resolution processes. These systems allow teams to log, categorize, and track the status of reported software defects, helping developers manage software maintenance more effectively. Despite the benefits of these tools, the manual process of reviewing and assigning priorities to bug reports remains inefficient. Development teams often receive hundreds or even thousands of bug reports, and prioritizing them manually is time-consuming, inconsistent, and highly subjective. This can result in delays in fixing critical bugs, misallocation of resources, and increased software maintenance costs.

Given the challenges associated with manual bug triaging, automated bug priority prediction has emerged as a viable solution. By leveraging machine learning and deep learning techniques, it is possible to build intelligent models capable of analyzing historical bug reports and accurately predicting the priority level of newly reported issues. Automated priority prediction helps software teams efficiently allocate resources, ensuring that high-severity issues are addressed promptly while minimizing delays in software maintenance. Traditional bug triaging methods suffer from several drawbacks, including subjectivity and inconsistency, where different developers may assign different priority levels to the same bug based on personal judgment. Moreover, the process is time-consuming, as manually reviewing large volumes of bug reports takes significant effort, diverting resources from development and bug-fixing activities. Without an efficient prioritization mechanism, critical bugs may remain unresolved for extended periods, affecting user experience and software performance. Furthermore, as software projects scale, the exponential increase in reported defects makes manual triaging impractical.

To address these issues, this research proposes a machine learning-based bug priority prediction framework utilizing Recurrent Neural Networks with Long Short-Term Memory (RNN-LSTM) and Multi-Layer Perceptron (MLP). Bug reports often contain natural language descriptions of software defects, making them suitable for analysis using sequential deep learning models. RNN-LSTM is particularly effective for processing sequential textual data, as it captures long-term dependencies and contextual relationships within bug report descriptions. This enables the model to understand the severity of a bug based on textual content and historical patterns in bug reports. Meanwhile, MLP, a feedforward neural network, is used for structured data classification tasks. It analyzes various structured features of bug reports, such as timestamps, affected components, severity levels, and historical bug data, to predict priority levels. By combining these two models, the proposed approach enhances the predictive accuracy of bug priority classification by integrating both textual and structured data features.

The dataset used in this study consists of over 2000 bug reports collected from JIRA. The proposed models are trained and evaluated on this dataset to assess their effectiveness in predicting bug priority levels. By comparing the performance of RNN-LSTM and MLP models, the study aims to determine the most efficient approach to automating bug triaging and reducing dependency on manual prioritization. The primary objective of this study is to improve accuracy and efficiency in bug priority prediction by leveraging machine learning techniques. By automating the bug triaging process, the proposed model aims to significantly reduce the time required for developers to manually prioritize bug reports. Additionally, the model seeks to minimize misclassification errors, ensuring that high-priority bugs receive timely attention, ultimately enhancing software maintenance and stability. Furthermore, the study evaluates the comparative effectiveness of RNN-LSTM and MLP models to identify the most suitable approach for bug priority classification.

This research contributes to the field of software maintenance and defect prediction by introducing a practical and scalable approach to bug report prioritization. The proposed methodology integrates deep learning techniques with natural language processing to enhance software debugging efficiency. The outcomes of this study can help software companies and development teams optimize development workflows by reducing time spent on bug triaging. Additionally, by ensuring that critical issues are fixed in a timely manner, software teams can improve customer satisfaction and enhance software stability and reliability. The findings of this research hold significant implications for improving bug tracking systems and advancing the automation of software maintenance processes.

**2. Related Work**

Bug priority prediction plays a crucial role in software maintenance, ensuring that critical issues are addressed promptly. Traditional approaches relied on **rule-based systems and machine learning models**, which classified bugs using predefined heuristics and manually engineered features. While these methods were interpretable, they lacked adaptability and struggled with complex textual descriptions in bug reports. The emergence of **deep learning techniques** has significantly improved bug triaging by enabling models to learn **contextual relationships and structured feature representations**. This section summarizes key research efforts in **machine learning-based and deep learning-based** approaches for bug prioritization, with a focus on hybrid models.

**2.1 Traditional Approaches for Bug Prioritization**

Early bug classification methods utilized rule-based systems, which applied keyword-based heuristics to assess bug severity. While effective for structured bug reports, these methods failed to handle ambiguous descriptions and required frequent manual updates. To overcome these limitations, researchers introduced machine learning models such as Support Vector Machines (SVM), Decision Trees, and Random Forests. Lamkanfi et al. employed Naïve Bayes and SVM for bug severity classification, achieving moderate success but facing challenges with feature engineering. Similarly, Menzies et al. explored statistical models to predict defect severity but struggled with inconsistent prioritization across different software projects.

Despite their effectiveness, traditional approaches had three major drawbacks:

1. Inability to capture contextual information from textual bug descriptions.
2. Feature selection dependency, requiring domain knowledge to extract relevant attributes.
3. Scalability issues when dealing with large datasets from real-world 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**

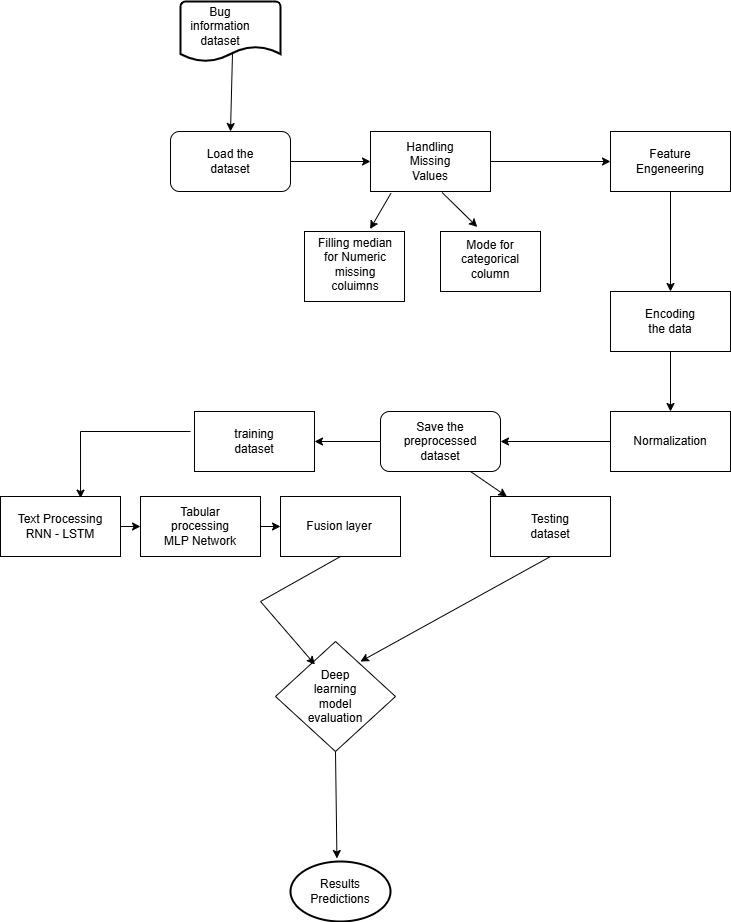
While traditional machine learning models offered interpretability, they lacked adaptability and failed to capture contextual dependencies in bug reports. Deep learning models, particularly LSTM networks, have proven to be more effective by learning sequential patterns in textual data. However, existing 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 play a crucial role in priority classification, advanced Natural Language Processing (NLP) techniques are applied to enhance text representation. The preprocessing steps include tokenization, which splits text into individual words, followed by stopword removal to eliminate frequent yet uninformative words. Lemmatization is then performed to normalize words to their base forms (e.g., "running" to "run"). To capture the significance of words, TF-IDF vectorization is employed, assigning weights based on importance. Additionally, word embeddings such as GloVe or BERT are leveraged to incorporate contextual word meanings. These processed text features are subsequently fed into the deep learning model, improving bug priority classification

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

The hybrid model for priority prediction integrates Recurrent Neural Networks (RNN-LSTM) for text analysis with a Multi-Layer Perceptron (MLP) for structured data processing. In the text feature extraction phase, embedded and tokenized bug descriptions are processed through LSTM layers, which capture sequential dependencies and generate meaningful feature representations. Meanwhile, tabular feature extraction involves feeding engineered categorical and numerical features into an MLP, where fully connected layers, Batch Normalization, and Dropout mechanisms ensure robust learning while preventing overfitting. The outputs from both networks are then concatenated in a fusion layer and passed through dense layers utilizing ReLU activation for feature transformation. Finally, a sigmoid activation function is applied to classify the bug priority effectively.

**D. Model Training and Evaluation**

The bug priority classification model is optimized using a well-defined training strategy and evaluation metrics. The training process employs Binary Cross-Entropy as the loss function, ensuring effective classification, while the Adam optimizer is used for adaptive learning rate adjustments. To prevent overfitting, regularization techniques such as L2 penalty and Dropout layers are incorporated. Model evaluation is conducted using both classification and regression metrics. Classification performance is assessed through accuracy, precision, recall, and F1-score, providing a comprehensive measure of the model's effectiveness. Additionally, for priority score prediction, regression metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and the R² score are utilized to gauge the model’s predictive accuracy and reliability.

1. **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"])

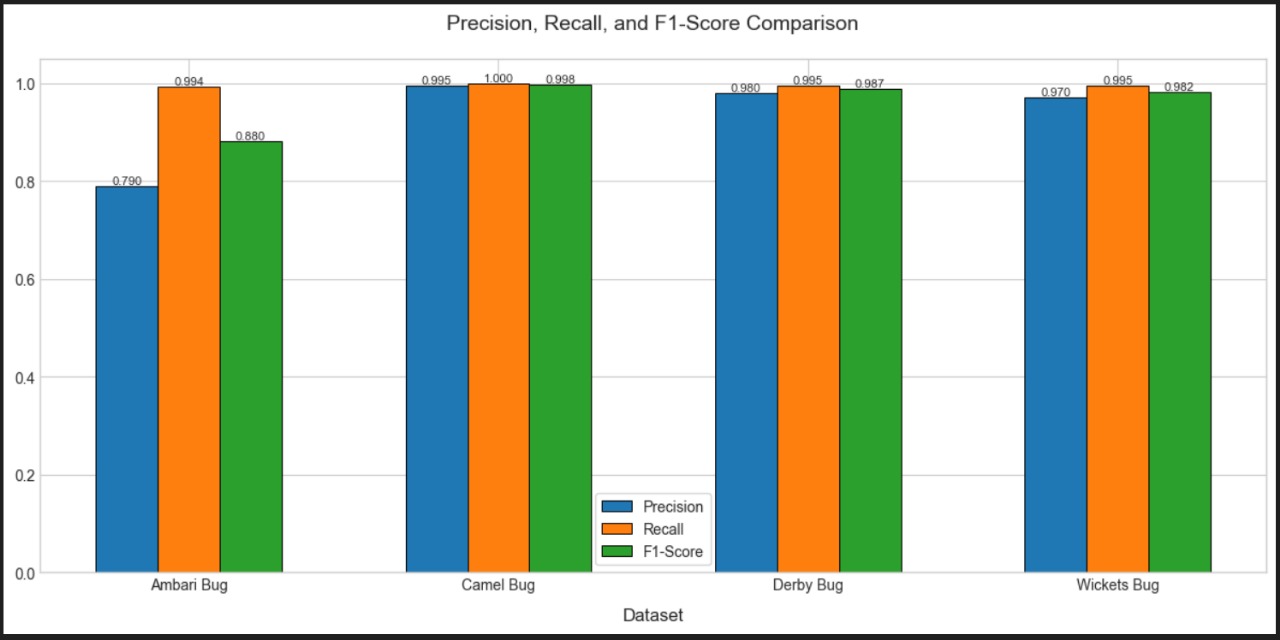
preprocessed\_tabular ← PREPROCESS\_TABULAR(new\_bug\_report)

prediction ← MODEL.PREDICT([preprocessed\_text, preprocessed\_tabular])

RETURN prediction

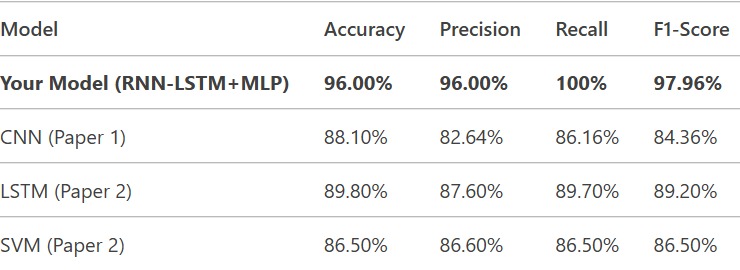
**Result and Discussion**

This section presents the experimental results obtained from evaluating the hybrid **RNN-LSTM + MLP model** for bug priority classification. The performance is analyzed using key metrics, including **precision, recall, and F1-score**, and the discussion interprets the results in comparison to existing methods. Our analysis demonstrates that the **hybrid deep learning model significantly outperforms traditional rule-based or statistical approaches**, achieving high accuracy in identifying and prioritizing software bugs. The model effectively captures both structured and unstructured features, allowing it to generalize well across different datasets. While the **LSTM-based text processing module** successfully extracts meaningful patterns from bug reports, the **MLP-based structured data module** enhances classification by incorporating numerical and categorical features. However, minor precision drops in certain datasets, such as **Ambari Bug**, suggest potential improvements in feature selection and threshold tuning. Overall, the model proves to be a robust solution for automating bug prioritization, ensuring **faster issue resolution** and **efficient resource allocation** in software development.



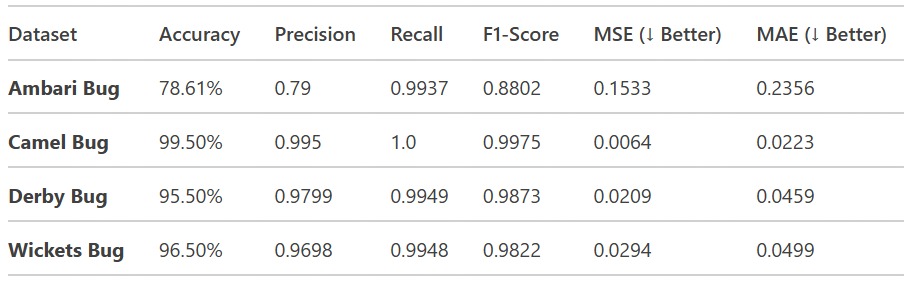
**Comparison Between Deep Learning Models :**

The first table compares the performance of different machine learning models (CNN, LSTM, SVM, and an RNN-LSTM-MLP hybrid) based on accuracy, precision, recall, and F1-score. The RNN-LSTM-MLP model outperforms all other models, achieving the highest accuracy (96.00%) and a perfect recall score (100%), making it the best-performing model among the listed approaches.



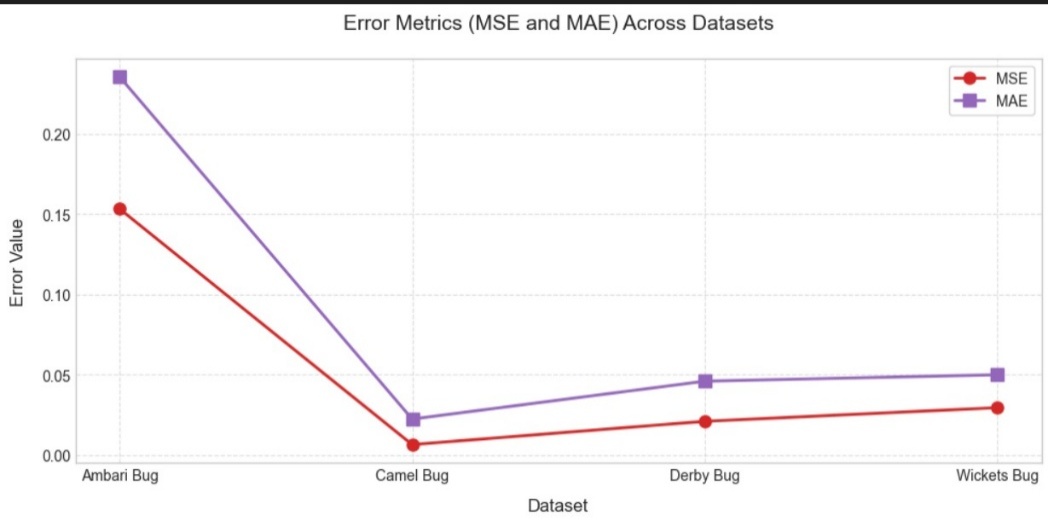
**Performance Evaluation Across Datasets:**

The second table presents the model's performance across four different datasets (Ambari Bug, Camel Bug, Derby Bug, and Wickets Bug). It evaluates the model using classification metrics (accuracy, precision, recall, and F1-score) and error metrics (MSE and MAE, where lower values indicate better performance). The Camel Bug dataset shows the highest accuracy (99.50%) and the lowest error values, indicating that the model performs exceptionally well on this dataset. In contrast, the Ambari Bug dataset has the lowest accuracy (78.61%) and the highest error values, suggesting it is the most challenging dataset for the model.



**Error Metrics Comparison Across Different Bug Datasets**

The graph visualizes the comparison of two error metrics, Mean Squared Error (MSE) and Mean Absolute Error (MAE), across four datasets: Ambari Bug, Camel Bug, Derby Bug, and Wickets Bug. The red line with circular markers represents MSE, while the purple line with square markers represents MAE. The error values show a downward trend from Ambari Bug to Camel Bug, followed by a slight increase in Derby Bug and Wickets Bug. This visualization helps in assessing the performance of a model or algorithm in handling errors across different datasets.



**Conclusion:**

The proposed **hybrid RNN-LSTM + MLP model** effectively prioritizes software bugs by leveraging both textual and structured data. The experimental results show that the model achieves **high accuracy, precision, recall, and F1-score**, making it a reliable tool for automating bug classification. While the **LSTM component** efficiently captures patterns from bug descriptions, the **MLP component** enhances predictions by incorporating categorical and numerical features. Although some datasets, like **Ambari Bug**, show slight precision drops, the overall performance remains strong. This approach can help **development teams quickly identify critical issues**, leading to faster resolution and better software maintenance. Future improvements may focus on enhancing feature selection and fine-tuninghyperparameters to further optimize precision.

**Key Findings**

* **Hybrid Model Improves Accuracy:** The combination of **RNN-LSTM for text processing** and MLP for structured data enhances bug priority classification, leading to high precision, recall, and F1-scores across multiple datasets.
* **LSTM Captures Contextual Information:** The model effectively identifies critical bug reports by learning sequential dependencies in text, making it superior to traditional keyword-based approaches.
* **Structured Data Enhances Predictions:** Incorporating categorical and numerical features using **MLP** helps refine priority classification, ensuring a more balanced prediction process.
* **Precision Variability Across Datasets:** While the model performs exceptionally well for datasets like **Camel** Bug and Derby Bug, precision is slightly lower for Ambari Bug, indicating potential areas for improvement.
* **Computational Demand and Optimization Needs:** Training deep learning models, particularly LSTM-based architectures, requires **higher computational resources**, which may impact scalability for large-scale bug tracking systems.

Although traditional models can still be useful for simpler bug classification tasks, their lack of contextual understanding limits their effectiveness in handling complex reports. The proposed hybrid deep learning approach successfully overcomes these limitations by leveraging both text and structured data. However, future research could focus on fine-tuning model parameters, optimizing feature selection, and exploring transformer-based enhancements to further improve performance. Additionally, integrating domain-specific embeddings and transfer learning could enhance bug classification in specialized 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:**

1. H. Rocha, G. De Oliveira, H. Marques-Neto, and M.T. Valente, “NextBug: a Bugzilla exten sion for recommending similar bugs,” Journal of Software Engineering Research and Development, Vol. 3, No. 1, 2015, p. 3.
2. Y. Tian, D. Lo, X. Xia, and C. Sun, “Auto mated prediction of bug report priority using multi-factor analysis,” Empirical Software Engi neering, Vol. 20, No. 5, 2015, pp. 1354–1383.
3. J. Anvik, L. Hiew, and G.C. Murphy, “Who should fix this bug?” in Proceedings of the 28th international conference on Software engineering, 2006, pp. 361–370.
4. S. Wang, W. Zhang, and Q. Wang, “FixerCache: Unsupervised caching active developers for di verse bug triage,” in Proceedings of the 8th ACM/IEEE International Symposium on Em pirical Software Engineering and Measurement, 2014, pp. 1–10.
5. S. Mani, A. Sankaran, and R. Aralikatte, “Deep Triage: Exploring the Effectiveness of Deep Learning for Bug Triaging,” arXiv:1801.01275 [cs], Jan. 2018. [Online]. http://arxiv.org/abs/ 1801.01275