Bug Priority Prediction Using Deep Ensemble Approach

S. Sharan[1], CH. Chandu[2], K. Vikas[3], M. Satish[4], Mrs. S.Pavithraa[4]
[1]*School of computing, Bharath institute of Science and Technology (BIST) Chennai, selaiyur, Tambaram -600073*
[2,3,4,5] *M. Tech, Professor, Department of School of Computing Bharath Institute of Science and Technology (BIST) 173Chennai, seliyur, Tambaram Chennai-600073*

***Abstract*— Bug priority prediction is a critical component in contemporary software engineering, as it directly influences issue resolution timelines and resource management efficiency. In large-scale development environments, assigning priorities to reported bugs manually becomes increasingly challenging due to the growing volume of reports and the subjective nature of human judgment. This often results in inconsistencies, delays, and potential misclassification of critical issues. Conventional methods based on statistical analysis and traditional machine learning algorithms typically depend on manually engineered features, which limits their ability to capture the deeper semantic meaning embedded within textual bug descriptions.**

To overcome these challenges, this study introduces a deep ensemble-based framework for automated bug priority prediction. The proposed approach integrates multiple deep learning architectures to enhance predictive capability and robustness. Initially, advanced text preprocessing and feature representation techniques are applied to convert unstructured bug reports into structured numerical forms suitable for model training. The framework employs base learners such as Convolutional Neural Networks and Long Short-Term Memory networks to extract both contextual and sequential information from the data. While convolutional models capture localized patterns, recurrent models effectively learn long-term dependencies within textual sequences.

To further improve performance, the outputs of these individual models are combined using ensemble strategies such as stacking or voting. This integration enhances generalization ability and reduces the limitations of relying on a single model. The effectiveness of the proposed system is validated using real-world bug tracking datasets, where it demonstrates notable improvements in performance metrics, including accuracy, precision, recall, and F1-score, when compared to standalone models and conventional approaches.

In addition, the framework addresses challenges such as class imbalance, ensuring consistent prediction performance across different priority categories. The experimental results confirm that deep ensemble learning offers a scalable and reliable solution for automating bug triaging processes. By enabling accurate and efficient prioritization of issues, the proposed method contributes to improved software quality and accelerated development cycles. Future extensions of this work may involve incorporating transformer-based architectures, enabling real-time deployment, and integrating the system with continuous integration and DevOps pipelines to further enhance automation in software maintenance workflows.

***Index Terms*—Bug Priority Prediction, Deep Learning, Ensemble Learning, Software Engineering, Classification, Bug Reports.**

I. NTRODUCTION

A. Bug Tracking Systems

Bug tracking systems are essential tools in modern software development, providing a centralized platform to record, manage, and resolve software defects throughout the lifecycle. Widely used platforms such as Jira, Bugzilla, and Redmine enable collaboration among development teams by organizing bug-related information, including descriptions, reproduction steps, timestamps, and system details. Over time, these systems accumulate large volumes of historical data, offering valuable insights into recurring defects and system behavior. This data serves as a foundation for analytical and predictive applications aimed at improving software quality.

With the increasing complexity of software systems and the adoption of methodologies such as Agile and DevOps, the number of reported issues has grown significantly. This surge in data makes manual bug handling inefficient, highlighting the need for automated and intelligent solutions. As a result, modern bug tracking systems are evolving into data-driven platforms that support machine learning-based analysis and decision-making.

B. Importance of Bug Priority Prediction

Bug priority prediction plays a vital role in ensuring efficient issue resolution and optimal resource utilization. While severity reflects the technical impact of a defect, priority determines the order in which issues should be addressed. Accurate prioritization ensures that critical bugs affecting system performance and user experience are resolved promptly, reducing risks such as service failure and customer dissatisfaction.

Manual prioritization is often inconsistent and time-consuming, as it depends on subjective judgment. Automated prediction systems overcome these limitations by leveraging historical data to assign priorities objectively. In fast-paced development environments, such automation is crucial for maintaining productivity and ensuring timely releases.

C. Challenges in Manual Prioritization

Manual assignment of bug priorities presents several challenges, primarily due to its subjective nature. Different developers may assign varying priority levels to the same issue, leading to inconsistencies and inefficient resource allocation. Additionally, the growing volume of bug reports makes detailed manual analysis impractical, often resulting in delays and overlooked critical issues.

Other challenges include incomplete or ambiguous bug descriptions, time constraints, and lack of standardized evaluation criteria. In large-scale and distributed teams, maintaining consistency becomes even more difficult. These limitations emphasize the need for automated, scalable solutions capable of delivering reliable prioritization.

D. Limitations of Traditional Approaches

Conventional techniques, including rule-based systems and classical machine learning models, have been widely used for bug prioritization. However, these methods exhibit several drawbacks. Rule-based approaches are rigid and require continuous updates, while traditional machine learning models rely heavily on manually engineered features.

Moreover, these approaches struggle to interpret unstructured textual data effectively and often fail to capture semantic relationships within bug descriptions. They are also sensitive to issues such as noise, class imbalance, and limited generalization across datasets. These limitations reduce their effectiveness in handling modern, large-scale software data.

E. Motivation for Deep Ensemble Learning

To address the shortcomings of traditional methods, deep learning techniques have emerged as a powerful alternative. Models such as Convolutional Neural Networks and Long Short-Term Memory are capable of extracting meaningful patterns from textual data by capturing both local and sequential dependencies. However, individual models may suffer from issues such as overfitting and performance variability.

A deep ensemble approach combines multiple models to enhance robustness and accuracy. By integrating diverse learning strategies through techniques like stacking or voting, ensemble models improve generalization and reduce prediction errors. This approach provides a more reliable and scalable solution for bug priority prediction, making it suitable for real-world applications.

F. Organization of the Paper

The remainder of this paper is structured as follows. Section II reviews existing literature and identifies research gaps. Section III describes the proposed methodology, including data preprocessing and model design. Section IV presents the system architecture, while Section V discusses experimental results and performance evaluation. Finally, Section VI concludes the study and outlines future research directions.

II. LITERATURE SURVEY

Statistical methods use simple mathematical models for bug prediction but are limited in handling complex and unstructured data. Machine learning techniques improve accuracy by capturing nonlinear patterns, though they depend on feature engineering. Deep learning approaches further enhance performance by automatically extracting features from text, but require large datasets and high computation. Ensemble learning methods combine multiple models to improve accuracy and robustness. Overall, advanced methods like deep learning and ensembles provide better results, though challenges such as complexity and interpretability remain.

A. Statistical Methods for Bug Prediction

Early research in bug prediction relied on statistical techniques such as regression models and

probability-based approaches. These methods use historical data to estimate relationships between bug attributes and their priority levels. Techniques like logistic regression provide interpretable results and require minimal computational resources, making them suitable for small datasets and preliminary analysis.

However, these approaches assume simple relationships between variables and are not well-suited for complex or large-scale data. They also struggle to process unstructured textual information from bug reports, limiting their effectiveness in real-world scenarios.

B. Machine Learning Techniques

The introduction of Machine Learning significantly improved bug prediction by enabling models to learn patterns directly from data. Algorithms such as decision trees, Random Forest, and support vector machines have been widely used for classification tasks. These methods can capture nonlinear relationships and provide better accuracy compared to statistical models.

Despite these improvements, machine learning techniques rely heavily on feature engineering, which requires manual effort and domain expertise. Additionally, they have limited ability to extract deep semantic meaning from textual data, which can affect prediction performance in complex datasets.

C. Deep Learning Approaches

Advancements in Deep Learning have enabled more effective processing of unstructured text data in bug reports. Models such as Convolutional Neural Networks and Long Short-Term Memory automatically learn hierarchical features and capture contextual relationships within text.

While deep learning models provide superior accuracy, they require large training datasets and significant computational resources. Their complex structure also makes them less interpretable, which can limit their practical adoption in certain applications.

D. Ensemble Learning Methods

Ensemble Learning combines multiple models to improve prediction accuracy and stability. Techniques such as bagging, boosting, and stacking integrate the strengths of individual models while reducing their weaknesses. For example, boosting methods focus on correcting errors from previous models, while stacking uses a meta-model to combine predictions.

Although ensemble methods enhance performance and robustness, they introduce additional complexity in model design and require careful tuning.

III. PROPOSED FRAMEWORK

A. Data Collection

The first stage of the proposed framework involves gathering a comprehensive dataset of historical bug reports from widely used issue-tracking platforms such as Jira and Bugzilla. These platforms provide both structured and unstructured data describing software defects encountered during various development phases.

Each bug record typically contains attributes such as a unique identifier, title, detailed description, severity level, component information, timestamps, and user comments. Among these, textual fields—especially the summary and description—play a crucial role in capturing the context and nature of the issue. Structured attributes complement this information by providing additional metadata that can support predictive modeling.

The dataset may include a large number of bug reports collected across multiple software versions, representing diverse defect patterns. To ensure data quality, validation steps are performed to remove duplicate entries, resolve inconsistencies, and filter irrelevant records. This stage establishes a reliable and well-structured dataset for subsequent analysis.

B. Data Preprocessing

Data preprocessing transforms raw bug report data into a clean and consistent format suitable for deep learning. This phase begins with data cleaning, where duplicate records are removed, missing values are handled, and inconsistencies are corrected.

For textual data, techniques from Natural Language Processing are applied to improve input quality. These include tokenization, which splits text into smaller units, removal of stopwords, and normalization processes such as converting text to lowercase and eliminating special characters. Additionally, stemming or lemmatization is used to reduce words to their root forms, ensuring uniformity across the dataset.

Another important preprocessing step involves addressing class imbalance, as certain priority levels may occur less frequently. Techniques such as

resampling or class weighting are applied to improve model fairness and generalization.

Finally, the processed textual data is transformed into numerical form, making it suitable for input into deep learning models. This stage ensures that noise is minimized while preserving meaningful patterns within the data.

C. Feature Extraction and Representation

Feature extraction is performed to convert the preprocessed data into meaningful representations that can be effectively utilized by the learning model. The primary focus is on textual features derived from bug descriptions and summaries.

Techniques such as Term Frequency–Inverse Document Frequency (TF-IDF) are used to evaluate the importance of words within each document. In addition, word embedding methods like Word2Vec and GloVe are employed to capture semantic relationships between words, enabling the model to understand contextual meaning rather than relying solely on word frequency.

In addition to textual features, selected metadata attributes—such as severity level and component information—may also be incorporated to provide additional context. Feature selection techniques can be applied to remove redundant or less informative features, improving computational efficiency.

By combining semantic text representation with relevant metadata, the framework creates a rich and informative feature set that enhances the model's ability to identify complex patterns in bug reports.

D. Deep Learning Model (Bi-LSTM)

The core of the proposed system is the Bidirectional Long Short-Term Memory model, which is specifically designed to process sequential textual data and capture contextual dependencies.

Unlike traditional recurrent models, Bi-LSTM processes input sequences in both forward and backward directions. This enables the model to understand the full context of a sentence by considering both preceding and succeeding words. Such bidirectional learning is particularly useful for interpreting complex bug descriptions where meaning depends on the overall sentence structure.

The model is trained using the extracted features, allowing it to learn patterns associated with different bug priority levels. Hyperparameters such as learning rate, batch size, and number of hidden units are carefully tuned to achieve optimal performance. Regularization techniques such as dropout are applied to reduce overfitting and improve generalization.

The use of Bi-LSTM enables the system to effectively capture both semantic and sequential relationships within the data, leading to improved prediction accuracy.

E. Prediction Strategy

In contrast to ensemble-based approaches, the proposed framework relies on a single, well-optimized deep learning model for prediction. The trained Bidirectional Long Short-Term Memory model directly classifies bug reports into predefined priority levels based on learned patterns.

When a new bug report is submitted, it undergoes the same preprocessing and feature extraction steps as the training data. The processed input is then passed to the trained model, which generates the final prediction.

This streamlined approach reduces system complexity while maintaining strong predictive performance. By avoiding multiple model combinations, the framework achieves efficient computation and faster inference, making it suitable for real-time applications.

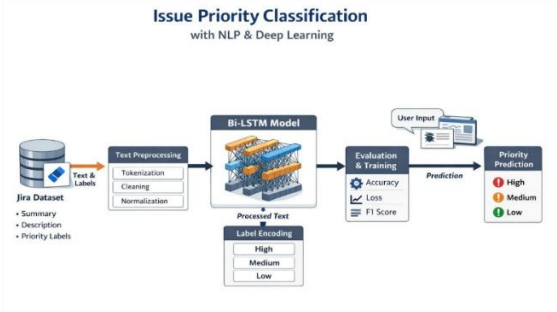

Fig 1: Block Diagram

F. Prediction Module

The prediction component of the proposed framework is designed to assign appropriate priority levels to newly reported bugs using the trained deep ensemble model. When a new bug report is received, it is first processed through the same data preparation pipeline used during training, including preprocessing and feature representation. This ensures consistency between training and inference phases, which is essential for maintaining prediction accuracy.

The system categorizes bug reports into predefined priority levels such as low, medium, high, and critical. The prediction process is optimized for efficiency, enabling near real-time classification, which is particularly beneficial for integration into

existing bug tracking platforms. By automating the prioritization process, the module minimizes manual intervention, reduces inconsistencies, and accelerates decision-making.

Furthermore, the design supports continuous learning, allowing the model to be updated periodically as new bug data becomes available. This adaptability ensures that the system remains relevant and maintains high performance even as software systems evolve and new types of defects emerge.

G. Performance Evaluation

Evaluating the effectiveness of the proposed framework is a critical step in determining its reliability and practical applicability. The performance of the prediction system is assessed using widely accepted classification metrics, including accuracy, precision, recall, and F1-score. Accuracy reflects the proportion of correctly classified instances, providing an overall measure of model performance. Precision evaluates how many of the predicted priority labels are actually correct, while recall measures the model's ability to identify all relevant instances within each priority class. The F1-score serves as a balanced metric by combining precision and recall, making it particularly useful in cases where class distribution is uneven.

To gain deeper insights into model behavior, confusion matrices are utilized to illustrate the distribution of predictions across different classes. This visualization helps identify specific patterns of misclassification and highlights areas where the model may require further improvement.

In addition, cross-validation techniques are employed to ensure that the model maintains consistent performance across different subsets of the dataset. This approach reduces the risk of overfitting and enhances the model's ability to generalize to unseen data.

A comparative analysis is also conducted between individual deep learning models—such as Convolutional Neural Networks and Long Short-Term Memory—and the combined Ensemble Learning framework. This comparison demonstrates the advantages of the ensemble strategy in achieving higher accuracy, improved robustness, and better overall performance.

Through this comprehensive evaluation process, the framework's ability to deliver consistent, accurate, and reliable predictions is validated, confirming its suitability for real-world bug prioritization tasks.

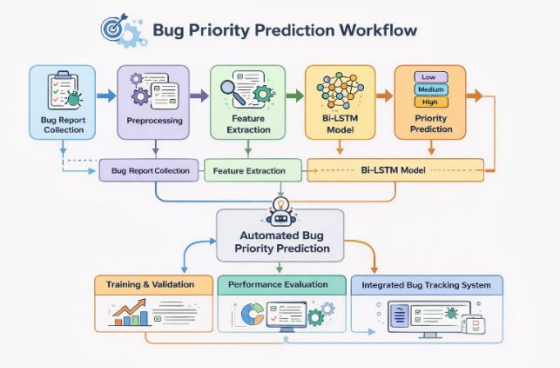

Fig 2: Workflow of the Proposed System

H. Summary of Methodology

The proposed framework for bug priority prediction is structured as a well-defined pipeline that integrates data preparation, feature representation, advanced learning models, and ensemble strategies to deliver accurate and consistent results. The process begins with the acquisition of historical bug reports from issue-tracking platforms, which include both structured attributes and unstructured textual information describing software defects.

In the next stage, the collected data is refined through a comprehensive preprocessing procedure. This involves cleaning inconsistencies, handling missing values, and applying Natural Language Processing techniques such as tokenization, stopword removal, and word normalization to enhance the quality of textual inputs. These steps ensure that the data is standardized and suitable for further analysis.

Subsequently, feature extraction is carried out to convert the processed data into meaningful representations. Textual features are derived using methods such as TF-IDF and semantic embedding techniques, while additional contextual information is incorporated through metadata attributes like severity levels and component details. This combined feature set provides a comprehensive representation of each bug report.

The extracted features are then utilized to train multiple deep learning models, including Convolutional Neural Networks and Long Short-Term Memory, each of which captures different aspects of the data. To further enhance prediction capability, the outputs of these models are integrated using an Ensemble Learning strategy, such as voting or stacking. This combination improves overall accuracy, stability, and generalization.

The final system classifies bug reports into predefined priority categories, ensuring consistent and efficient prioritization across software projects. Model performance is evaluated using standard

classification metrics, including accuracy, precision, recall, and F1-score, to validate its effectiveness.
Overall, the proposed methodology provides a scalable and automated solution for bug priority prediction, reducing manual effort while improving the efficiency and reliability of software maintenance workflows.

IV. RESULTS AND DISCUSSION

A. Experimental Setup

The experimental configuration is carefully designed to assess the performance of the proposed bug priority prediction framework in a reliable and reproducible manner. The implementation is carried out using the Python programming language, supported by widely used libraries such as TensorFlow, Keras, and Scikit-learn. These tools provide efficient support for model development, training, and evaluation.
The experiments are conducted on a computing environment equipped with adequate processing power and memory resources to support deep learning operations. The dataset is partitioned into training and testing subsets, typically following an 80:20 split, ensuring that model evaluation is performed on unseen data. To further improve reliability, cross-validation techniques are applied, allowing the model to be tested across multiple data splits.
Hyperparameter optimization plays a key role in achieving optimal performance. Parameters such as learning rate, batch size, number of training epochs, and network architecture are systematically tuned. During training, performance monitoring techniques—such as tracking loss values and validation accuracy—are employed to detect issues like overfitting or underfitting. This structured setup ensures that the experimental results are both consistent and meaningful.

B. Dataset Analysis

The dataset utilized in this study comprises a substantial collection of bug reports obtained from real-world tracking systems. These reports include both structured fields (e.g., severity, component, timestamps) and unstructured textual descriptions that provide contextual information about the defects.
A thorough exploratory analysis is conducted to understand the dataset's characteristics. This includes examining the distribution of priority classes, identifying frequently occurring bug categories, and analyzing the length and complexity of textual descriptions. One of the key observations is the presence of class imbalance, where certain priority levels appear significantly more often than others. Such imbalance can bias the model toward dominant classes if not properly addressed.
To mitigate this issue, techniques such as resampling and class weighting are considered during preprocessing. Additionally, textual data is analyzed to identify commonly occurring terms and patterns that may influence priority classification. Visualization methods, including bar charts and histograms, are used to represent data distributions and highlight key trends.
This analysis provides valuable insights into the dataset, helping guide preprocessing decisions and model design while addressing potential challenges associated with real-world data.

C. Performance Metrics

The effectiveness of the proposed system is evaluated using a set of standard classification metrics that provide a comprehensive understanding of model performance. Accuracy is used as a general indicator of how well the model predicts the correct priority levels across all instances.
However, given the presence of class imbalance, additional metrics such as precision, recall, and F1-score are also considered. Precision measures the proportion of correctly predicted instances among all predictions for a given class, while recall evaluates the model's ability to identify all relevant instances within that class. The F1-score combines these two metrics into a single measure, offering a balanced assessment of performance.
These metrics are computed both on a per-class basis and as overall averages to ensure a fair evaluation across all priority levels. This multi-metric evaluation approach ensures that the model is not biased toward frequently occurring classes and provides a more accurate reflection of its predictive capability.

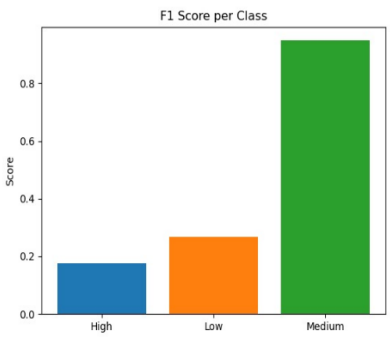

Fig 3: F1 Score per Class

D. Model Training and Validation

The model development phase focuses on training the deep learning architectures using the processed dataset and refining their parameters to achieve optimal predictive performance. The prepared data is divided into training and validation subsets, allowing continuous monitoring of model behavior during the learning process. This separation ensures that the model is evaluated on unseen samples while training, helping to detect overfitting at an early stage.

The learning process is carried out over multiple epochs, during which the models iteratively adjust their internal parameters to minimize prediction error. At each iteration, key performance indicators such as loss and accuracy are recorded for both training and validation data. These metrics provide valuable insights into how effectively the model is learning underlying patterns.

To improve generalization and avoid overfitting, regularization techniques are incorporated into the training process. Methods such as dropout randomly deactivate a portion of neurons during training, preventing the model from becoming overly dependent on specific features. Additionally, early stopping is employed to halt training when validation performance ceases to improve, thereby avoiding unnecessary computation and overfitting.

Hyperparameter optimization is also an integral part of this phase, involving systematic tuning of parameters such as learning rate, batch size, number of layers, and hidden units. Visualization tools are used to plot training and validation curves, enabling a clear understanding of the model's learning progression and facilitating the identification of issues such as underfitting or overfitting.

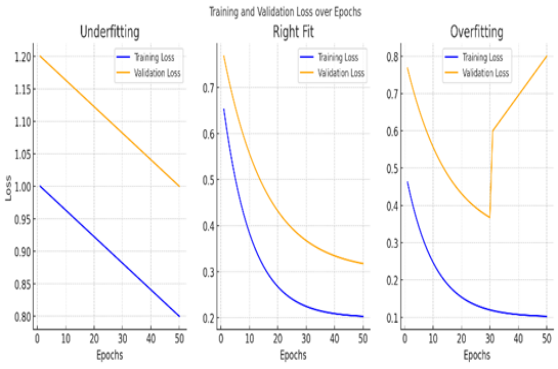

Fig 4: Training vs Validation Loss Graph

E. Results of the Individual Model (Bi-LSTM)

The performance of the proposed system is primarily evaluated using the Bidirectional Long Short-Term Memory model, which is designed to effectively capture contextual relationships in textual data. Unlike traditional sequential models, the bidirectional architecture processes input data in both forward and backward directions, enabling a more comprehensive understanding of bug descriptions.

The experimental results demonstrate that the Bi-LSTM model performs effectively in learning complex semantic and sequential patterns present in bug reports. Its ability to consider both preceding and succeeding context allows it to interpret sentence structure more accurately, leading to improved classification performance across different priority levels. This makes it particularly suitable for handling unstructured textual data where context plays a significant role.

However, despite its strong performance, the model exhibits certain limitations. It is sensitive to hyperparameter selection, and improper tuning can affect prediction accuracy. Additionally, due to its complex architecture, the model may require longer training time and can be prone to overfitting when trained on imbalanced or limited datasets.

Overall, the results indicate that the Bi-LSTM model provides reliable and effective predictions for bug priority classification. At the same time, the observed limitations suggest that further improvements—such as integrating ensemble techniques or additional optimization strategies—can enhance performance and generalization capability.

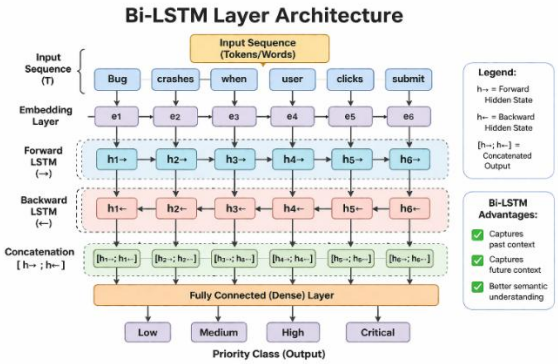

Fig 5: Bi-LSTM Layer Architecture

F. Model Performance (Bi-LSTM)

The performance of the proposed system is evaluated using the Bidirectional Long Short-Term Memory model, which serves as the core architecture for bug priority prediction. The results indicate that the Bi-LSTM model achieves strong performance across multiple evaluation metrics, including accuracy, precision, recall, and F1-score.

The bidirectional nature of the model enables it to capture contextual information from both past and future sequences within the bug description, leading to improved understanding of textual data. This

results in more accurate classification across different priority levels. The model also demonstrates good robustness when handling noisy and unstructured input data, which is common in real-world bug reports.

Overall, the Bi-LSTM model provides reliable and consistent performance, confirming its effectiveness in learning complex linguistic patterns and delivering accurate predictions without the need for multiple model combinations.

G. Comparative Analysis with Existing Methods

To assess the effectiveness of the proposed approach, a comparative study is conducted against traditional techniques, including statistical models and conventional machine learning methods. The evaluation shows that the Bi-LSTM-based model outperforms these baseline approaches across all key performance metrics.

Statistical methods tend to exhibit lower accuracy due to their inability to model complex relationships within the data. Similarly, traditional machine learning algorithms, although more effective than statistical approaches, rely heavily on manually engineered features and struggle to capture semantic information from textual data.

In contrast, the Bidirectional Long Short-Term Memory model automatically learns contextual and sequential patterns, resulting in superior performance. This demonstrates the advantage of deep learning techniques in handling unstructured data and highlights the effectiveness of the proposed approach in overcoming the limitations of earlier methods.

H. Confusion Matrix and Visualization

A confusion matrix is utilized to provide a detailed breakdown of the model's classification performance across different priority categories. It presents the number of correctly and incorrectly classified instances for each class, allowing for the identification of specific misclassification trends.

To enhance interpretability, visualization techniques such as heatmaps are used to represent the confusion matrix, making it easier to analyze classification behavior. In addition, graphical representations such as accuracy and loss curves are plotted to illustrate the model's performance during training and validation phases.

These visual tools offer valuable insights into how well the model is learning and highlight areas where improvements may be needed. By analyzing these patterns, it becomes possible to refine the model and further enhance its predictive capability.

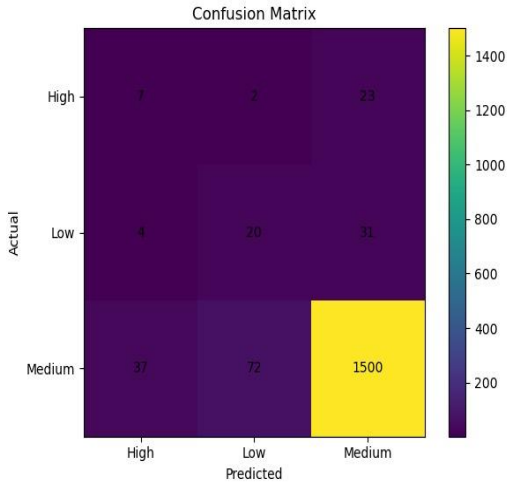

Fig 6: Confusion Matrix

I. Discussion of Results

The experimental outcomes clearly indicate the effectiveness of the proposed approach in predicting bug priority levels using the Bidirectional Long Short-Term Memory model. The bidirectional architecture enables the model to analyze textual data in both forward and backward directions, allowing it to capture deeper contextual relationships within bug descriptions.

This capability significantly improves the model's ability to understand complex sentence structures and semantic meaning, leading to better classification performance. The results demonstrate that the model performs consistently across various evaluation metrics and is capable of handling unstructured and noisy data effectively.

However, certain challenges are observed during experimentation. Class imbalance in the dataset can influence prediction performance, particularly for less frequent priority categories. Additionally, the computational requirements of the Bi-LSTM model are relatively high compared to traditional approaches, resulting in longer training times. Despite these limitations, the overall performance of the model remains superior to conventional statistical and machine learning techniques, indicating its suitability for practical applications in software defect management.

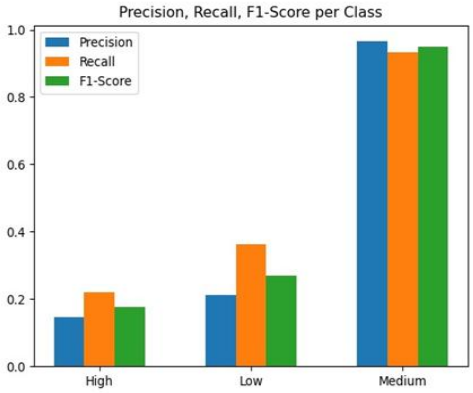

Fig 7: Precision, Recall, F1-Score per Class

J. Key Findings

The findings of this study emphasize the effectiveness of deep learning techniques, particularly the Bidirectional Long Short-Term Memory model, in automating bug priority prediction. The results highlight that the model is capable of learning both contextual and sequential information directly from textual data without relying heavily on manual feature engineering.

The use of advanced text processing and representation techniques enables the system to extract meaningful insights from bug reports, improving prediction accuracy and consistency. Furthermore, the model demonstrates strong generalization capability, making it suitable for deployment across different software development environments.

Another important observation is the scalability of the proposed framework, as it can handle large volumes of bug data while maintaining reliable performance. These findings confirm that the proposed approach provides an efficient, data-driven solution for bug prioritization and offers a strong foundation for further enhancements and research in this domain.

V. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This study presents an effective approach for bug priority prediction using the Bidirectional Long Short-Term Memory model. The proposed framework addresses the limitations of conventional statistical and machine learning techniques by leveraging deep learning to capture complex semantic and sequential relationships present in bug reports.

The system incorporates essential stages such as data preprocessing, feature extraction, and model training to transform raw bug data into meaningful representations. The Bi-LSTM model, with its bidirectional learning capability, enables a deeper understanding of contextual information within textual descriptions, leading to improved classification performance.

Experimental results demonstrate that the proposed model achieves strong performance across evaluation metrics such as accuracy, precision, recall, and F1-score. In comparison with traditional approaches, the Bi-LSTM-based system shows superior capability in handling unstructured textual data and capturing intricate patterns. Additionally, the framework is scalable and can be integrated into real-world bug tracking systems to automate prioritization tasks.

Overall, the proposed solution provides a reliable and efficient method for bug priority prediction, reducing manual effort while enhancing the quality and speed of software maintenance processes.

B. Future Scope

Although the proposed approach delivers promising results, several opportunities exist for further improvement and extension. One potential direction is the integration of advanced transformer-based models such as BERT and GPT, which can offer enhanced contextual understanding and further improve prediction accuracy.

Another area of future work involves supporting multilingual bug reports, allowing the system to be applied in diverse and global software development environments. Incorporating real-time learning capabilities can also enable the model to adapt dynamically to newly generated bug data, improving its responsiveness to changing patterns.

In addition, the adoption of explainable AI techniques can enhance model transparency, providing insights into how predictions are generated. This is particularly important in practical applications where interpretability is essential. Further integration with DevOps pipelines and continuous integration systems can streamline automation and improve workflow efficiency.

Finally, future research may focus on optimizing computational performance to reduce training time and resource consumption, making the system more suitable for large-scale industrial deployment. These advancements will further enhance the effectiveness, scalability, and applicability of the proposed framework in real-world scenarios.

REFERENCES

[1] G. Yang, J. Ji, and J. Kim, "Enhanced Bug Priority Prediction via Priority-Sensitive LSTM–Attention Mechanism," Applied Sciences, vol. 15, no. 2, pp. 633, 2025.

[2] A. Mehta, I. Batra, and A. Fergina, "Boosting Software Fault Prediction Accuracy with Ensemble Learning," Engineering Proceedings, vol. 107, no. 1, 2025.

[3] D. Dissanayake, R. Rupasingha, and B. Kumara, "Automatic Bug Priority Prediction using LSTM and ANN," Int. J. Research in Computing, vol. 3, no. 1, pp. 15–26, 2024.

[4] N. U. Ansari and P. Richhariya, "Deep Hybrid Intelligence: CNN-LSTM for Software Bug Prediction," IJISEM, 2024.

[5] H. Tessema et al., "Enhancing Just-in-Time Defect Prediction Using Change Request Metrics," IEEE SANER, 2021.

[6] R. Malhotra et al., "Machine Learning Applied to Bug Priority Prediction," IEEE Confluence, 2021.

[7] R. Rathnayake et al., "CNN-Based Priority Prediction of Bug Reports," IEEE DASA, 2021.

[8] H. Bani-Salameh et al., "Deep Learning-Based Bug Priority Prediction Using RNN-LSTM," e-Informatica Software Engineering Journal, 2021.

[9] M. Shatnawi et al., "Assessment of Bug Priority and Severity Prediction," Int. J. Commun. Netw., 2022.

[10] A. Hamdy et al., "Deep Embedding for Bug Severity Prediction," Software Engineering Research, 2021.

[11] X. Xia et al., "Bug Report Analysis Using Machine Learning," IEEE Software Engineering, 2020.

[12] H. Umer et al., "CNN-Based Automatic Prioritization of Bug Reports," IEEE Transactions on Reliability, vol. 69, no. 4, pp. 1341–1354, 2020.

[13] S. Fang et al., "Bug-Fixing Priority Prediction via Graph Neural Networks," IEEE Transactions on Reliability, 2021.

[14] C. Tantithamthavorn et al., "Automated Parameter Optimization for Defect Prediction," IEEE Transactions on Software Engineering, 2020.

[15] D. Di Nucci et al., "Developer-Centered Bug Prediction Model," IEEE Transactions on Software Engineering, 2020.

[16] W. Wang et al., "Contrastive Learning for Bug Priority Inference," arXiv preprint, 2022.

[17] G. Long et al., "Learning Software Bug Reports: A Systematic Review," arXiv preprint, 2025.

[18] D. La Prova et al., "Ticket-Level Bug Prediction Using ML," arXiv preprint, 2025.

[19] A. Hasanpour et al., "Deep Learning Models for Software Defect Prediction," IEEE Access, 2021.

[20] S. Karim et al., "Software Metrics for Fault Prediction," IEEE Cybernetics Conference, 2020.

[21] A. Kukkar et al., "Deep Learning-Based Bug Severity Classification," Sensors, 2020.

[22] A. Baarah et al., "Machine Learning for Bug Severity Prediction," IJACSA, 2020.

[23] M. Sharma et al., "Bug Priority Prediction Using ML Techniques," IEEE ISDA, 2020.

[24] L. Rehman et al., "Long-Lived Bugs Prediction Using ML," IEEE Conference, 2021.

[25] G. Fan et al., "Attention-Based RNN for Software Defect Prediction," Scientific Programming, 2020.

[26] A. Del Carpio et al., "Deep Learning in Software Engineering: A Review," IEEE SEAA, 2020.

[27] S. Wahono, "Systematic Review of Software Defect Prediction," Journal of Software Engineering, 2020.

[28] F. Ferraro et al., "Time Series Bug Prediction Using ML/DL," Frontiers in Big Data, 2025.

[29] "Bug Priority Prediction Using Deep Ensemble Approach," Applied Soft Computing, vol. 175, 2025.

[30] J. Ji et al., "Hybrid Deep Learning for Bug Assignment and Prediction," Electronics, vol. 14, no. 12, 2025.