Bug Severity Prediction using Deep Learning

1st Ahmed Shoaib
Computer Science
Habib University
Karachi, Pakistan
as07313@st.habib.edu.pk

2nd Asad Muhammad
Computer Science
Habib University
Karachi, Pakistan
am07127@st.habib.edu.pk

3rd Asadullah Chaudhary
Computer Science
Habib University
Karachi, Pakistan
ac07408@st.haib.edu.pk

***Abstract*—This report evaluates and presents the final report of our study on utilizing a deep neural network for automated bug severity prediction. Bug severity assignment in bug reporting is a crucial aspect of software development processes which allows the development and quality assurance teams to identify and resolve critical issues in a timely and efficient manner. Traditional bug reporting processes often rely on a manual bug severity assignment which is prone to subjectivity and and lacks standardization. To address this issue, numerous studies have been conducted to exploit the power of deep learning techniques in the bug severity prediction of software bug reports. Our goal is to utilise deep learning models like RoBERTa, ALBERT (A Lite BERT), DistilBERT, and DeBERTa to perform this task and perform a comparative analysis of their relative accuracies.**

***Index Terms*—bug prediction, severity levels, deep learning, transformers**

I. Introduction

Software system performance is critically dependent on reliability and the absence of bugs,faults, and flaws,which, if let unchecked, can render a system unusable.It is more crucial than ever to identify software flaws accurately and promptly as our reliance on automated and effective software systems increases. Designed to minimize human intervention, embedded software systems have been instrumental in speeding up procedures and lowering error rates in a variety of digital and electronic systems. However, these benefits cannot be fully realized without effective mechanisms for detecting and assessing software bugs.

Software bug detectors have been a huge help in determining the severity of bugs and setting priorities for fixing them. The term "bug severity" describes how a defect affects a software system's functionality; these are frequently divided into four levels: low, medium, high, and critical [6]. However, manual bug severity assignment is prone to human error and subjectivity, leading to misclassification of bug importance. This challenge has driven interest in the development of automated techniques for predicting bug severity levels.

The primary objectives of this literature review are to comprehensively examine existing approaches to bug report severity prediction, evaluating their effectiveness and identifying areas for improvement. Specifically, this review aims to provide an in-depth analysis of current machine learning and deep learning techniques utilised in severity prediction, assessing their performance in terms of accuracy, precision, and recall. Additionally, this study seeks to investigate the impact of dataset characteristics, quality, and size on severity prediction models, as well as explore practical implications and industry adoption of these systems. By synthesising the existing body of knowledge, this review aims to identify research gaps and outline future directions for advancing bug report severity prediction research.

II. Research Questions

The research question we are addressing is **"How do modern deep learning models perform in predicting the severity level of bug reports based on their short and long descriptions?"** Our problem statement can be formulated as follows:

"Given the short description and long description of a bug report, predict its severity level".

We will be exploring and training different deep learning models. The inputs to the model will be the short description and long description of a bug report and the output will be a class label denoting the severity class it belongs to from six different classes (0-6). The motivation of this study stems from the fact that although numerous studies have been conducted that explore the effectiveness of traditional deep learning models such as convolutional neural networks in predicting the bug severity levels of bug reports, some modern transformer based models such as RoBERTa, ALBERT, DistilBERT, and DeBERTa remain unexplored for this particular task. Thus, our research question investigates how well state-of-the-art Transformer-based models perform compared to each other, focusing on critical classification metrics such as precision, recall, and F1-score to determine the most efficient model. By addressing these questions, this research aims to provide insights into how different deep learning architectures can be utilized to predict the bug severity, making it more accurate and efficient.

III. Literature Review

Previous studies have extensively utilized convolutional neural networks to predict bug severity levels. One such study [1] highlighted the impact of using multi-aspect features like the quality aspect, content-aspect and reporter aspect and feeding them into a convolutional neural network to predict the severity level class. They performed experiments on Eclipse

and Mozilla data and found their model to beat the state-of-the-art CNN model in terms of average Accuracy, Precision, Recall and F1-measure by 1.83% , 0.46%, 3.23% and 1.72% respectively.

Another study [2] used topic based feature algorithm and LSTM-CNN algorithms to first classify bug reports by topic-based severity and then extract features from the severity of each topic. The proposed approach was more efficient than other baselines and the F-measure statistic was recorded at 90.62 % for Mozilla.

One notable contribution made by [3] was developing a BERT-based severity prediction model (BERT-SBR) for mobile application maintenance. The study crucially highlights that though the approaches like CNNs and LSTMs have demonstrated better performance over traditional methods but they lack contextual understanding limiting their effective in accurately classifying bug severity. The BERT-SBR model applies a fine-tuned BERT classifier to predict bug severity in mobile application maintenance. Using sentiment analysis and BERT tokenization, the model processes bug reports and produces. word embeddings which are then used by a fine-tuned BERT classifier to predict severity levels. The study reveals a significant improvement in performance metrics with a 40.43 % increase in accuracy, 67.78 % in precision, 40.71 % in recall, and 58.14 % in F1-score. outperforming other deep learning classifiers. This method demonstrates the ability of deep learning models, specifically attention-based architectures like BERT, to capture complex textual patterns in bug reports, resulting in more accurate severity predictions.

Another study [4]. Bug assignment refers to the process of automatically identifying the most suitable developer to address a reported bug, which is a crucial step in the bug management workflow. The evaluation is performed in large-scale open source project due to the sheer volume of bug reports and the varying expertise of developers. The authors approach bug assignment as a text classification task, where the textual description of the bug serves as the input, and the potential fixer is the output label. It investigates 35 combinations of five word embedding models (Word2Vec, GloVe, NextBug, ELMo. and BERT) and seven deep learning classification techniques (TextCNN, LSTM, Bi-LSTM, LSTM with attention, BI-LSTM with attention, MLP and Naive Bayes). The evaluation across datasets (Eclipse, JDT, GCC, Firefox, and a cross project dataset) shows that Bi-LSTM with attention and ELMo outperforms other combinations in terms of top-k accuracy and Mean Recriprocal Rank.Additionally, the study highlights importance of the bug report description over the summary in improving assignment accuracy and emphasizes the critical role of fine-tuning word embedding models based on training corpus. [4]

However, the current research has focused primarily on models like BERT [3], CNN [2], and LSTM [4], leaving room for exploration of more advanced and efficient variations such as RoBERTa, ALBERT, DistilBERT, and DeBERTa. These models offer improvements in terms of efficiency, scalability, and resource usage, while maintaining high levels of accuracy. Additionally, the use of attention-based models remains under explored in bug severity prediction, providing a fertile area for further research. Including these models in future studies can yield valuable insights by comparing their performance against existing methods.

Year	Model	Accuracy	Dataset
2021 [1]	MASP- CNNs	1.83 % better than conventional CNNs	Mozilla and Eclipse Projects
2021 [5]	K-Nearest Neighbour (KNN)	70.7 %	Kaggle Dataset
2022 [2]	CNN-LSTM	90.62 % (F-Score Measure)	Mozilla and Eclipse Projects
2023 [3]	BERT-SBR	Accuracy-91.13 %, Precision-91.02 %, Recall-91.13 %, and F-Score Measure - 91.03 %	HuggingFace, SenitWordNet
2024 [4]	Word2Vec, GloVe, NextBug, ELMo, and BERT	The top-k (k=1, 5, 10) accuracy and Mean Reciprocal Rank (MRR) were reported for each model.	Eclipse JDT, GCC, Firefox

Fig. 1. A Comparison of different models

IV. Dataset

We utilize the dataset created by Gomez et al. [5], created to go along with their publication called 'Long Lived Bug Prediction' (2021). Their dataset contains 50,618 bug reports extracted from Bugzilla, pertaining to 6 different open source projects (shown in Fig.1) Fig.1 provides an overview of the dataset characteristics of the dataset as reported by the authors [5].

FLOSS projects used in our research.

Project	Number of bugs	Observation period	
		From	To
Eclipse	9724	2001-10-10	2018-01-30
Freedesktop	7644	2003-02-05	2019-08-15
GCC	9946	1999-02-26	2018-01-31
Gnome	7772	1999-01-02	2018-01-24
Mozilla	9945	1998-04-15	2014-04-22
WineHQ	6037	2000-09-27	2018-04-17

Fig. 2. Dataset Characteristics

Furthermore, Table 1 identifies the features each bug report in the dataset contains. There are other features for each bug report in the original dataset Table 1 shows the ones that are relevant to our project. This paper aims to use the short and long description of a bug to predict what it's severity level should be, and thus we will be focusing on these three features in the dataset.

Fig.3 provides a high level view of what our model that we are building would look like. The multi-classification model would take in the short and long description (as a concatenated string) of a bug report and output the severity level of the bug report. We will be using a pre-trained BERT model to extract features from the text data and then feed these features into a deep neural network to predict the severity level of the bug report. We will be evaluating the multiple models on our dataset (using F1 score and accuracy) namely BERT,

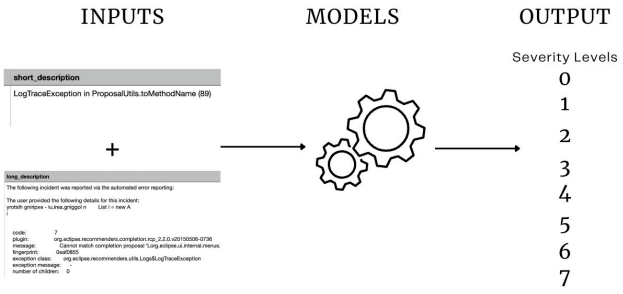
TABLE I
DATA FEATURES AND DESCRIPTION

Feature	Description
Bug ID	Unique identifier for each bug report
Creation Date	Date when the bug report was created
Short description	Short description of the bug report
Long description	Detailed description of the bug report
Resolution Date	Date when the bug report was resolved
Severity Code	Severity level of the bug report
Severity Description	Description of the severity level

s

RoBERTa, ALBERT , DistilBERT, and DeBERTa and record as well as compare their respective accuracy's in performing this task. Utilizing Google Collab's free tier to train our model, we will evaluate our results, and we hope to get 90%+ accuracy on at least a few models. As we can see in Fig 1, our current dataset contains data till 2018 and thus scraping more data from Bugzilla (for the period 2018 to 2024) will be considered if we cannot achieve our desired accuracy.

Fig. 3. Model High-Level Diagram



V. MODELS AND EXPERIMENTS

A. Experimental Setup

This study evaluates the performance of various state-of-the-art transformer-based models for severity classification of bug reports. The models analyzed include:

- BERT Base Uncased
- RoBERTa
- DistilBERT
- DeBERTa
- ALBERT

1) Data Preprocessing: The dataset was preprocessed by merging the short and long descriptions of each bug report into a unified text representation, ensuring comprehensive contextual information. Severity levels were encoded as categorical labels ranging from 0 to 6, facilitating multi-class classification.

2) Model Configuration: The methodology employed for each model was standardized as follows:

1) **Tokenization:** Pre-trained model-specific tokenizers were utilized to convert the text into numerical input sequences.
2) **Input Preparation:**
   - Maximum sequence length was set to 128 tokens.
   - Padding and truncation were applied to ensure uniform input dimensions.

B. Training Methodology

The training process followed a structured approach to ensure consistency across models:

- **Dataset Split:** The dataset was divided into 80% training and 20% validation sets.
- **Batch Size:** 16
- **Learning Rate:** 1×10^{-6}
- **Number of Epochs:** 10
- **Optimization Algorithm:** AdamW optimizer
- **Loss Function:** Cross-Entropy Loss

C. Evaluation Metrics

To comprehensively assess model performance, the metrics used were validation accuracy, performance and F-1 score.

D. Hyperparameter Optimization

Hyperparameter tuning was conducted systematically using the Optuna library, leveraging advanced search strategies to optimize model performance.

1) Key Hyperparameters Tuned: The following hyperparameters were adjusted:

- Learning rate
- Batch size
- Model-specific architectural parameters

E. Computational Infrastructure

All experiments were conducted on a GPU-enabled computing environment to accelerate training and inference processes, ensuring efficient handling of computationally intensive tasks.

VI. RESULTS AND EVALUATION

Model	Validation Loss	Validation Accuracy	Validation F1 Score
BERT	0.5622	0.8468	0.7878
ALBERT	0.7875	0.7993	0.7731
RoBERTa	0.6632	0.8331	0.7573
DistilBERT	0.5989	0.8420	0.7747
DeBERTa	0.6305	0.8435	0.7715

A. Addressing Class Imbalance

The previous results indicated a class imbalance issue in the dataset, which likely impacted the model's performance. The classification report and accuracy scores revealed that the model underperformed on certain classes due to the disproportionate representation of samples in the training data. We saw that Class 2 made up 84% of our data while the other 6 classes accounted for 16%. To address this, the following techniques were implemented:

1) **Weighted Loss Function**: The standard cross-entropy loss function was modified to include class weights. These weights were computed based on the inverse

frequency of each class in the training data, effectively emphasizing the underrepresented classes during training.

2) **Weighted Random Sampler**: A `WeightedRandomSampler` was implemented to dynamically balance the training samples during each epoch. This ensured the model encountered more examples from the underrepresented classes, improving its ability to learn their patterns.
3) **Learning Rate Scheduler**: A `StepLR` scheduler was used to dynamically reduce the learning rate during training. This helped the model converge more effectively, particularly when dealing with imbalanced data.
4) **Early Stopping**: An early stopping mechanism was introduced to monitor validation loss and terminate training if performance plateaued. This prevented overfitting, especially on the majority classes.

By incorporating these techniques, the goal was to mitigate the effects of class imbalance and enhance the overall performance of the BERT-based classification model. The follwoing table shows our revised results:

Model	Validation Loss	Validation Accuracy	Validation F1 Score
BERT	1.0775	0.6162	0.6731
DistilBERT	1.0638	0.6012	0.6673
RoBERTa	1.0623	0.6119	0.6754

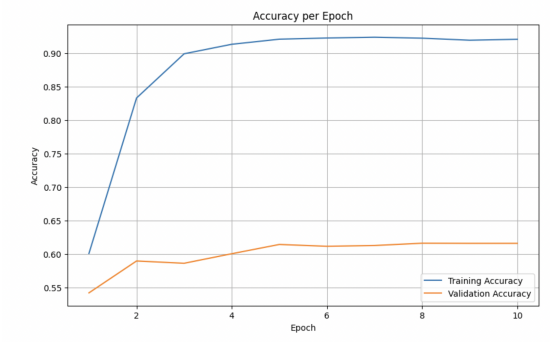


Fig. 4. BERT accuracy during training phase

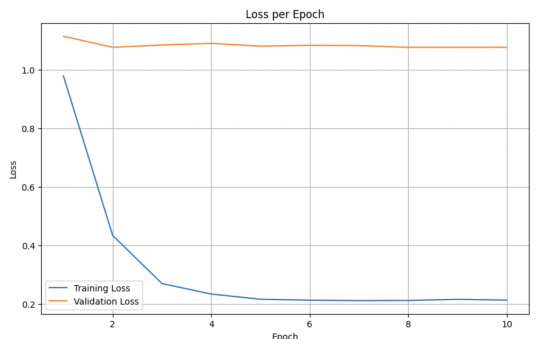


Fig. 5. BERT loss during training phase

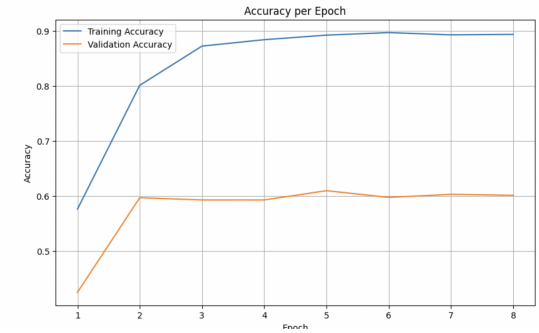


Fig. 6. DistilBERT accuracy during training phase

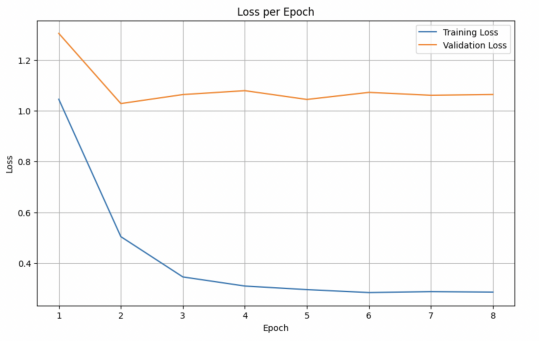


Fig. 7. DistilBERT loss during training phase

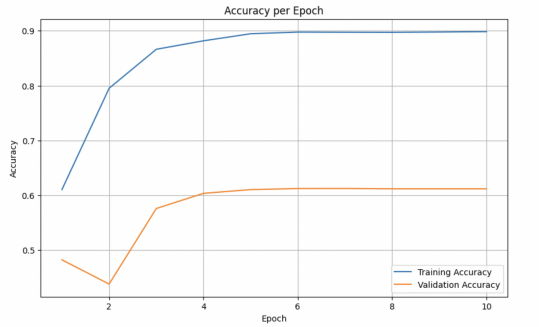


Fig. 8. RoBERTA accuracy during training

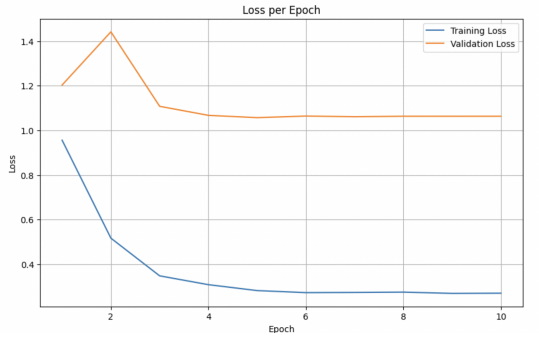


Fig. 9. RoBERTA loss during training

B. Discussion

Here we compare our results with other researches in the same domain. The study that was the motivational factor for our work on the long-lived bugs dataset was one of the prime benchmarks when evaluating results from our experiments. This study [5] had exclusively utilized machine learning algorithms as the basis of their comparisons and our aim was to expand upon it by including deep learning techniques to this work as well. Like us, they incurred class imbalance as well in the early stages of their study and took appropriate steps to rectify it. The technique used by them for this purpose was the Synthetic Minority Oversampling technique also known as SMOTE. This statistical technique uses new instances extracted from existing minority cases which are then used to increase the number of cases in the dataset in a balanced way. This study passed summary as well as descriptions from each bug report to the model and reported results for both cases.

Since our study exclusively utilized descriptions from bug reports as the input to our models, we will only be comparing the description input results for comparison purposes from the study. It is important to note that the study only reported balanced accuracy in its report. Before balancing, our transformer models consistently outperformed all the models reported in this study including simplified vector machines, K-nearest Neighbors and Random Forest Classifiers having an accuracy of around 84% as opposed to a 50% accuracy reported by conventional machine learning algorithms.

It is worth noting that our methodology to tackle the issue of class imbalance was different from the mentioned study, utilizing a weight random sampler and weighted loss functions. Post-balancing, we were able to report results for only three of our models due to time constraints which were BERT base-uncased, DistillBERT and RoBerta. Among them, DistilBERT's performance was the best registring an accuracy of 61.62%. This was better than the performance of naive bayes post-balancing from the referenced study which reported an accuracy of only 53.3%. Apart from that, all models consistently outperformed transformers with simplified vector machines achieving an impressive accuracy of 98.1%. This was surprising as another study [3] which utilized BERT for a similar severity prediction task on a publicly available dataset of app reviews from Hugging Face, consisting of 50,000 reviews with severity ratings ranging from 1 (most severe) to 5 (least severe), achieved commendable performance results of 91.13% for accuracy, 91.02%, 91.13% for Recall, and 91.03% for F-score measure.

C. Future Directions

Further analysis led us to the conclusion that the dataset they used was structured differently from ours and for a slightly different task which limits direct comparison. It is interesting to note that in their architecture, SentiWordNet was also used to compute the sentiment scores of bug reports (i.e. the feelings of user while writing the text which can range from negative to positive) and those sentiment scores along with the date of the report were passed as input to the model. Moreover, their approach included fine-tuning transformer models specifically for this task, an idea that could also be incorporated by us in future work. A key challenge in our study was addressing class imbalance, with approximately 80% of the dataset belonging to a single severity label. For future research, data augmentation strategies, such as adding labeled bug reports from Bugzilla or similar bug repositories to handle class imbalance issue. Moreover, diversifying the sources of bug reports would also help in increasing diversification of dataset that would ensure better representation across all severity label, thereby enhancing model performance.

By addressing these limitations and exploring the outlined directions, future studies can build upon the foundation laid by this study, contributing to the development of more reliable, scalable, and accurate systems for bug severity prediction.

REFERENCES

[1] Dao, A.-H.; Yang, C.-Z. Severity Prediction for Bug Reports Using Multi-Aspect Features: A Deep Learning Approach. Mathematics 2021, 9, 1644. https://doi.org/10.3390/math9141644

[2] J. Kim and G. Yang, "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm," in IEEE Access, vol. 10, pp. 94643-94651, 2022, doi: 10.1109/ACCESS.2022.3204689.

[3] Ali, Y. Xia, Q. Umer, and M. Osman, "BERT based severity prediction of bug reports for the maintenance of mobile applications," Journal of Systems and Software, vol. 208, p. 111898, Feb. 2024, doi: 10.1016/J.JSS.2023.111898.

[4] R. Wang, X. Ji, S. Xu, Y. Tian, S. Jiang, and R. Huang, "An empirical assessment of different word embedding and deep learning models for bug assignment," Journal of Systems and Software, vol. 210, p. 111961, Apr. 2024, doi: 10.1016/j.jss.2024.111961.

[5] Gomes, Luiz; Torres, Ricardo; Côrtes, Mario (2021), "A Dataset for Long-lived Bug Prediction in FLOSS ", Mendeley Data, V2, doi: 10.17632/v446tfssgj.2

[6] S. Ahlawat, "A Literature Review on Software Bug Severity", Ijicse, vol. 11, no. 1, pp. 8-12, Mar. 2024.