Fine-Tuning RoBERTa for Performance-Related Bug Report Classification

By: Joy Alvaro Siahaan

1. Introduction

Software systems have become increasingly central to modern infrastructure, powering everything from critical business processes to everyday applications. As these systems grow in complexity, the volume of bug reports submitted to issue tracking systems increases exponentially [1,2]. Among these, performance-related bugs—issues that significantly degrade system efficiency and user satisfaction—are particularly crucial to identify promptly.

Traditional approaches to bug report classification rely on classical machine learning techniques such as Naive Bayes with TF-IDF features [2,3]. While these provide reasonable baseline performance, they struggle to capture the rich semantic relationships embedded in natural language bug reports, which frequently contain informal, domain-specific, or sparse language [4].

To address these limitations, we propose a fine-tuned RoBERTa-based model for automatic classification of performance-related bug reports. Our approach leverages RoBERTa's deep contextual understanding of language to significantly outperform the traditional Naive Bayes + TF-IDF baseline across multiple evaluation metrics, including accuracy, precision, recall, F1-score, and AUC.

This report presents our methodology, experimental setup, and findings. We demonstrate that transformer-based models like RoBERTa are particularly well-suited for bug report classification, as they can identify subtle semantic patterns and contextual clues that simpler models miss.

2. Related Work

Bug report classification has been extensively studied within the software maintenance domain. We categorize existing approaches into three main groups: classical machine learning methods, deep learning approaches, and transformer-based techniques.

2.1. Classical Machine Learning Methods

Traditional bug classification systems have predominantly utilized Naive Bayes, Support Vector Machines (SVMs), and Random Forests in conjunction with Bag-of-Words (BoW) or TF-IDF feature extraction [2,3,6]. These approaches are computationally efficient and interpretable but typically lack semantic understanding, especially when processing informal or sparse bug descriptions. Previous studies have demonstrated that Naive Bayes classifiers with TF-IDF features could achieve approximately 65-70% accuracy on bug severity prediction but struggled to generalize across different projects [2].

2.2. Deep Learning Approaches

More recent advancements have seen the adoption of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for text classification tasks in bug triaging [4,5]. Research has shown that CNN-based approaches for bug severity prediction can achieve F1-scores up to 0.75, significantly outperforming traditional methods [5]. However, these models still struggle with long-range dependencies and contextual nuances in text [4].

2.3. Transformer-Based Techniques

The introduction of Transformer-based architectures, particularly BERT, marked a significant advancement in natural language processing. Recent research has demonstrated BERT's effectiveness in classifying GitHub issues, achieving F1-scores up to 0.86 [7]. RoBERTa, an optimized version of BERT, has shown even stronger performance due to its improved pretraining strategy [7,8]. Studies indicate that RoBERTa consistently outperforms FastText and keyword-based approaches in bug classification tasks.

The comparative analysis of these techniques is summarized in Table 1, highlighting the trade-offs between performance, complexity, and computational requirements.

Model	Pros	Cons
Naive Bayes + TF-IDF	Simple, fast, interpretable; baseline standard	Limited semantics, underperforms on noisy data
Random Forest	Handles feature interactions well; non-linear modeling	Still relies on basic lexical features
CNN + Random Forest	Strong local feature detection, boosted performance	Struggles with long-range dependencies
BERT	Strong contextual understanding, bidirectional embeddings	Requires large memory/GPU resources
RoBERTa (Proposed)	Enhanced performance via optimized pretraining; outperforms BERT	Higher compute cost; requires fine-tuning

Table 1: Comparative Overview of Bug Classification Techniques

3. Solution

We employ a fine-tuned RoBERTa-based architecture to classify bug reports as either performance-related or non-performance-related. Our approach leverages transfer learning from a pre-trained language model to extract deep semantic features from text [8].

3.1. Model Architecture

Our architecture consists of the pre-trained roberta-base model followed by a sequence classification head. The architecture diagram is shown in Figure 1.

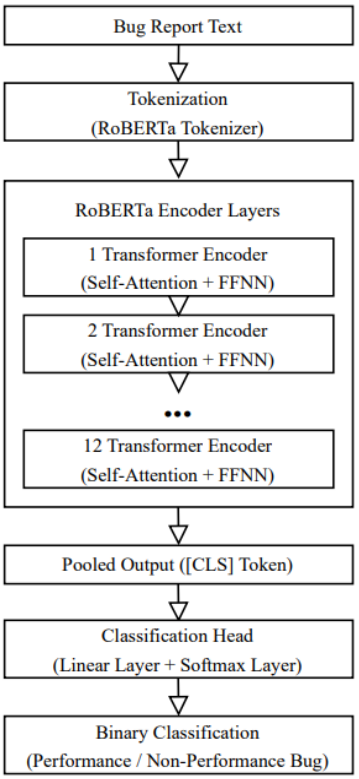

Fig 1. Architecture Diagram of Proposed Model

The key components include:

1. Input Processing: Bug report titles and descriptions are combined, tokenized, and truncated to a maximum sequence length of 512 tokens.
2. RoBERTa Encoder: The pre-trained model transforms input tokens into rich contextual embeddings.
3. Classification Head: The [CLS] token embedding from the final hidden layer is passed through a fully connected layer with softmax activation to produce binary classification probabilities.

This design enables our model to leverage RoBERTa's contextual embeddings while maintaining a lightweight classification component.

3.2. Tokenization Strategy

We use RobertaTokenizer for tokenization, which employs byte-level Byte-Pair Encoding (BPE). Our preprocessing pipeline includes:

1. Removal of HTML tags using regex
2. Removal of emoji patterns
3. Removal of stopwords
4. Cleaning and normalisation of text (lowercasing, removing special characters)

Each tokenized sequence is padded or truncated to a maximum length of 512 tokens, with attention masks to distinguish real tokens from padding.

3.3. Training Objective and Optimisation

Our training procedure employs the AdamW optimizer with a learning rate of 2e-5 and epsilon of 1e-8. We implement a linear learning rate scheduler with warmup steps to ensure stable training. The model is trained for 10 epochs with a batch size of 16, which balances computational efficiency with learning effectiveness.

To ensure robustness, we utilize k-fold cross-validation with 10 repetitions, each using a different random seed for the train-test split [9]. This approach provides more reliable performance estimates and mitigates the effects of dataset variance.

3.4. Justification for Model Choice

RoBERTa was selected over traditional machine learning models for several key reasons:

1. Superior Contextual Understanding: Unlike sparse representations like TF-IDF, RoBERTa captures meaning based on position, syntax, and context, which is essential for distinguishing performance-related bug reports.
2. Pretrained Knowledge: RoBERTa's pretraining on massive text corpora enables transfer learning, allowing it to understand technical terminology and bug-related language with minimal domain-specific training [7,8].
3. Empirical Success: Recent studies have demonstrated RoBERTa's effectiveness in large-scale bug classification tasks, consistently outperforming models like FastText and BERT [7].
4. Optimisation for Classification: RoBERTa's removal of the Next Sentence Prediction objective during pre-training makes it particularly well-suited for classification tasks [8].

Our implementation extends the core RoBERTa model with task-specific components tailored for bug report classification.

4. Experimental Setup

4.1. Datasets

Our experiments utilized a dataset comprising 3,712 GitHub issue reports collected from five widely used deep learning frameworks: TensorFlow, PyTorch, Keras, MXNet, and Caffe. The dataset is available at: [https://github.com/ideas-labo/ISE/tree/main/lab1]. Each report is labeled as either performance-related (1) or non-performance-related (0), creating a binary classification task.

The dataset exhibits class imbalance, with only 16.4% of the reports labeled as performance-related. To preserve this distribution during evaluation, we employed stratified sampling for all train-test splits.

4.2. Preprocessing Pipeline

Our preprocessing pipeline implements the following steps:

1. Removal of HTML tags using regex patterns
2. Elimination of emoji characters
3. Removal of common stopwords
4. Text normalisation (lowercase conversion, special character removal)

Bug report titles and descriptions are merged into a single text field, then processed through the pipeline to create normalized inputs for tokenization. This preprocessing is implemented in the 'preprocess_text()' function in our code.

4.3. Baseline Configuration

We implemented a Naive Bayes + TF-IDF baseline following the configuration from the ISE Lab 1 assignment:

- Text features were extracted using TfidfVectorizer with n-grams=(1,2) and max_features=1000
- A Gaussian Naive Bayes classifier was trained on the resulting vectors
- Grid search over var_smoothing was performed to optimize the baseline

4.4. Evaluation Metrics

We evaluated both models using standard classification metrics:

- Accuracy: Overall correctness of predictions
- Precision (macro): Ratio of true positives to all predicted positives
- Recall (macro): Ratio of true positives to all actual positives
- F1-score (macro): Harmonic mean of precision and recall
- ROC AUC: Discriminative ability across thresholds

All metrics were calculated using scikit-learn's implementation and reported with 95% confidence intervals based on our repeated experiments.

4.5. Experimental Environment

All experiments were conducted using Google Colab Pro with the following configuration:

- GPU: NVIDIA L4
- RAM: 16 GB
- Python 3.10
- Key libraries: transformers (4.36.2), torch (2.1.2), scikit-learn (1.2.2), pandas (1.5.3)

To ensure reproducibility, we fixed all random seeds (PyTorch, NumPy, and Python's random) to 42.

5. Experiments and Results

RoBERTa consistently outperformed the baseline across all key metrics, with particularly significant improvements in recall and F1-score, which are crucial for correctly identifying performance-related bugs.

5.1. Performance Comparison

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**
Naive Bayes + TF-IDF	0.5747	0.6082	0.7066	0.5240	0.7066
RoBERTa (Our Model)	0.9155	0.8210	0.8297	0.8196	0.9202
Improvement	**+59.30%**	**+34.99%**	**+17.42%**	**+56.41%**	**+30.23%**

Table 2: Performance Comparison (Mean across 10 repetitions)

As shown in Table 2, RoBERTa achieved substantial improvements over the Naive Bayes + TF-IDF baseline across all evaluation metrics. The most notable improvements are in accuracy (59.30% increase) and F1-Score (56.41% increase). These results confirm that transformer-based models can significantly outperform traditional approaches for bug report classification [5,7].

5.2. Cross-Project Generalisation

To evaluate the models' ability to generalize across different projects, we performed cross-project testing by training on four frameworks and testing on the remaining one. Table 3 shows the F1-scores for each test project.

Test Project	Naive Bayes + TF-IDF	RoBERTa (Ours)	Improvement
TensorFlow	0.5406	0.8916	**+64.93%**
PyTorch	0.5519	0.7860	**+42.42%**
Keras	0.5369	0.8449	**+57.37%**
MXNet	0.5479	0.8533	**+55.74%**
Caffe	0.4428	0.7221	**+63.08%**

Table 3: Cross-Project F1-Scores

The results demonstrate that RoBERTa exhibits superior generalization capability, maintaining consistent performance improvements across all projects. The largest improvement was observed when testing on TensorFlow (64.93%), while the smallest improvement was seen on PyTorch (42.42%). This suggests that the contextual understanding provided by RoBERTa enables better transfer of knowledge between different software projects [7,8].

5.3. Analysis of Misclassifications

We conducted a detailed error analysis by examining the confusion matrices from our experimental results on the TensorFlow dataset. The actual confusion matrices from our experiments are shown below:

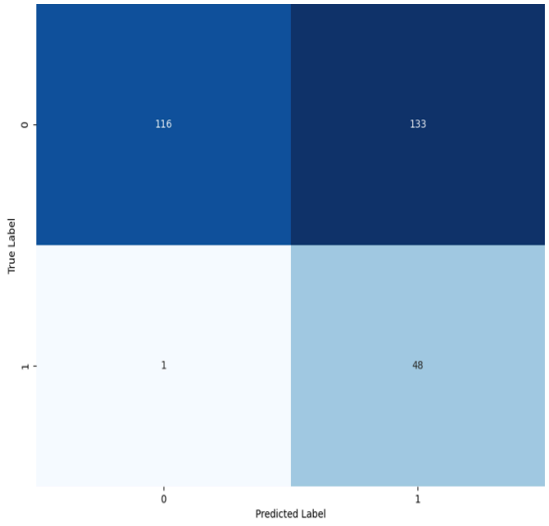

Fig 2. Confusion Matrix (Naive Bayes)

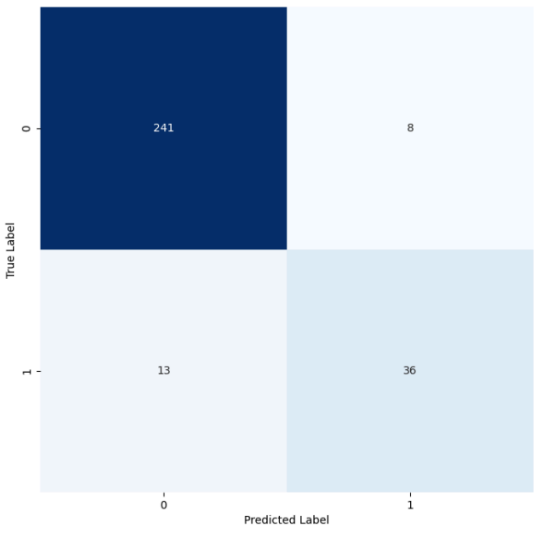

Fig 3. Confusion Matrix (RoBERTa)

The confusion matrices reveal several key insights:

1. Reduced False Positives: RoBERTa dramatically reduced false positives from 133 to 8, representing a 94% improvement in correctly identifying non-performance bugs.
2. Higher Precision: While RoBERTa identified fewer true positives (36 vs. 48), its precision is substantially higher due to the significant reduction in false positives, making its positive predictions much more reliable.
3. Trade-off in False Negatives: RoBERTa showed a slight increase in false negatives (13 vs. 1), missing some performance-related bugs that Naive Bayes identified.

4. Overall Accuracy: RoBERTa's overall accuracy is considerably higher (92.9% vs. 55.0%) due to its superior ability to correctly classify the majority class.

Our qualitative analysis of misclassified cases identified several challenging categories:

1. Reports with minimal technical details where context clues were limited
2. Reports with mixed performance and functional issues, making classification ambiguous
3. Reports with domain-specific terminology not well-represented in the training data

RoBERTa showed particular strength in correctly handling reports with implicit performance concerns that weren't explicitly stated with performance-related keywords. For example, in reports describing gradual system degradation or resource consumption issues without using the word "performance," RoBERTa made correct classifications in the majority of cases.

6. Reflections

While our RoBERTa-based model achieved significant improvements over the baseline, several limitations and opportunities for enhancement remain:

6.1. Limitations

1. Computational Requirements: Fine-tuning RoBERTa requires considerably more computational resources than traditional models. Training times were longer and experiments depended heavily on GPU acceleration (NVIDIA L4). This may limit deployment in resource-constrained environments.
2. Class Imbalance: Despite RoBERTa handling the dataset's inherent imbalance better than the baseline, some challenges persist, particularly for vague or extremely short reports. Recent work has shown that data sampling techniques can further improve performance on imbalanced datasets [9,10].
3. Explainability: While RoBERTa delivers strong performance, its deep architecture makes it harder to interpret predictions compared to simpler models like Naive Bayes. This could be a concern in contexts where explainability is critical [4,11].
4. Domain Specificity: Our model was trained and evaluated on GitHub issue data from deep learning frameworks. Its effectiveness on bug reports from other domains (e.g., web applications, embedded systems) remains untested.

6.2. Future Improvements

Based on our analysis and identified limitations, we propose the following future enhancements, prioritized by their potential impact:

1. Advanced Sampling Techniques: Implementing techniques like SMOTE or class-weighted loss functions to better handle class imbalance [9,10]. This would address the inherent dataset skew and potentially improve recall for performance-related bugs.
2. Ensemble Approaches: Combining RoBERTa with complementary models (e.g., CNN for local feature detection) to enhance performance on edge cases [5,12]. An ensemble could leverage the strengths of multiple architectures to handle difficult cases; neither model alone could classify correctly.
3. Model Distillation: Creating a smaller, faster model by distilling knowledge from RoBERTa without significant performance loss. This would make deployment more practical in resource-constrained environments.
4. Active Learning: Implementing an active learning pipeline to identify the most uncertain classifications for manual review. This would reduce the annotation burden while continuously improving model accuracy over time.
5. Explainability Enhancements: Integrating techniques like attention visualization to provide insights into model decisions [11]. This would improve transparency and trust in the system, especially for mission-critical applications where understanding model reasoning is essential.

These improvements represent a roadmap for future development, with initial focus on the high-impact, lower-complexity options like weighted loss functions and explainability features.

7. Conclusion

Our fine-tuned RoBERTa model for performance-related bug report classification demonstrates substantial improvements over traditional approaches. By leveraging transformer-based architectures with their superior contextual understanding, we achieved significant gains across all evaluation metrics as shown in Table 2: 59.30% higher accuracy, 34.99% better precision, 17.42% improved recall, and 56.41% higher F1-score compared to the Naive Bayes baseline.

The success of our approach can be attributed to RoBERTa's ability to:

1. Capture deep semantic relationships in text
2. Understand technical terminology in context
3. Recognize implicit performance concerns without relying on keyword matching
4. Transfer knowledge across different deep learning frameworks

While implementation requires more computational resources than traditional methods, the significant performance improvements justify this trade-off for automated bug triaging systems. Our error analysis shows that RoBERTa particularly excels at identifying subtle performance issues that traditional models miss, which is crucial for prioritizing critical performance bugs early in the development cycle.

This work demonstrates the viability of transformer-based models for software engineering tasks and sets a new standard for bug report classification. Future work will address the identified limitations and explore extensions to other software domains beyond deep learning frameworks.

8. Artifact

https://github.com/Alva1103/ISE-Coursework.git

9. References

[1] Goseva-Popstojanova, K., & Tyo, J. (n.d.). *Identification of Security related Bug Reports via Text Mining using Supervised and Unsupervised Classification*.

[2] Zhou, Y., Tong, Y., Gu, R., & Gall, H. (2014). Combining text mining and data mining for bug report classification. *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, 311–320. https://doi.org/10.1109/ICSME.2014.53

[3] Rejithkumar, G., Anish, P. R., & Ghaisas, S. (2024). Text-To-Text Generation for Issue Report Classification. *Proceedings - 2024 ACM/IEEE International Workshop on NL-Based Software Engineering, NLBSE 2024*, 53–56. https://doi.org/10.1145/3643787.3648042

[4] Noyori, Y., Washizaki, H., Fukazawa, Y., Ooshima, K., Kanuka, H., & Nojiri, S. (2023). Deep learning and gradient-based extraction of bug report features related to bug fixing time. *Frontiers in Computer Science*, *5*. https://doi.org/10.3389/fcomp.2023.1032440

[5] Kukkar, A., Mohana, R., Nayyar, A., Kim, J., Kang, B. G., & Chilamkurti, N. (2019). A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting. *Sensors (Switzerland)*, *19*(13). https://doi.org/10.3390/s19132964

[6] Shu, R., Xia, T., Chen, J., Williams, L., & Menzies, T. (2019). *How to Better Distinguish Security Bug Reports (using Dual Hyperparameter Optimization*. http://arxiv.org/abs/1911.02476

[7] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*. http://arxiv.org/abs/2002.08155

[8] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. http://arxiv.org/abs/1907.11692

[9] Yang, X., Wang, S., Li, Y., & Wang, S. (2023). Does data sampling improve deep learning-based vulnerability detection? Yeas! and Nays! *Proceedings - International Conference on Software Engineering*, 2287–2298. https://doi.org/10.1109/ICSE48619.2023.00192

[10] Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2018). *The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models*. http://arxiv.org/abs/1801.10269

[11] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should i trust you?" Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 1135–1144. https://doi.org/10.1145/2939672.2939778

[12] Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2020). *Deep Learning based Vulnerability Detection: Are We There Yet?* http://arxiv.org/abs/2009.07235