

A Contrastive Learning Approach to Bug Severity Classification with Large Language Model Embeddings

Mosarrat Rumman, Emon Roy, Anushka Zaman, Jeremy S. Bradbury
Ontario Tech University
Oshawa, ON, Canada

{mosarrat.rumman, emon.roy, anushka.zaman}@ontariotechu.net, jeremy.bradbury@ontariotechu.ca

Abstract—Automatically classifying bug severity helps reduce manual effort and improve response times in software maintenance. This study leverages Large Language Models (LLMs), specifically CodeBERT, to generate contextual embeddings of bug reports for automated severity classification. To enhance the quality of embeddings, we integrate Contrastive Learning, which structures the embedding space by bringing similar bug reports closer and pushing dissimilar reports apart. Our approach is evaluated on the NASA PITS and Mozilla datasets and compared against LLMs fine-tuned without contrastive learning as well as traditional embedding models like Doc2Vec. Our results demonstrate that contrastive learning consistently enhances performance, particularly on imbalanced and diverse datasets. Furthermore, the results demonstrate that LLMs excel in handling longer bug descriptions, while traditional embedding models like Doc2Vec are suitable for smaller, structured datasets.

Index Terms—bug severity, contrastive learning, large language models, natural language processing, software maintenance.

I. INTRODUCTION

Bug severity classification plays a critical role in software maintenance by helping developers triage and prioritize bug reports. Assigning the correct severity level to a given bug report ensures that critical issues impacting core functionality are fixed promptly [1]. Manual bug severity classification is challenging in large-scale software projects and can be inefficient, labor-intensive, and prone to inconsistencies. Alternatively, automatic bug severity classification can improve efficiency, ensure consistency, enhance scalability, and enable faster and more reliable software maintenance.

Automating bug severity classification involves analyzing bug reports and assigning them to predefined severity levels. Traditionally, bug severity classification has utilized rule-based approaches, keyword matching, and machine learning models trained on manually extracted features [2]. More recent approaches have been developed to improve performance using deep learning [3], graph attention networks (GATS) [4], and embedding techniques (e.g., Doc2Vec) [5]. These existing methods are often limited in their ability to generalize across projects with diverse bug reports.

In contrast, Large Language Models (LLMs) offer a more flexible and generalizable approach to bug severity classification. Unlike the models mentioned above, LLMs automatically learn contextual embeddings from bug descriptions, capturing semantic nuances without the need for handcrafted feature selection [6]. While fine-tuning LLMs for bug severity classification can improve performance, they still suffer from misclassification when overlapping representations where embeddings of different severity levels may be too similar. This problem is particularly evident in imbalanced datasets, where certain severity classes dominate, making it difficult for models to learn distinct patterns for underrepresented classes.

To enhance the benefits of LLMs and to better handle imbalanced datasets [7], we propose a contrastive learning-based approach for fine-tuning LLMs in bug severity classification¹. Contrastive learning enhances the embedding space by bringing similar bug reports closer together while pushing dissimilar reports apart, ensuring better differentiation between severity levels. By structuring the embeddings more effectively, this approach improves classification accuracy, robustness, and generalization, particularly in imbalanced and diverse datasets. In this research, a novel framework for bug severity classification is introduced, where a pre-trained LLM (CodeBERT) is fine-tuned using contrastive learning. To assess the benefits of our framework, we evaluate it using the NASA PITS dataset and the Mozilla Bug Reports dataset, and conduct a comparative analysis with LLMs trained without contrastive learning and traditional embedding models (Doc2Vec). Results demonstrate improvements in classification accuracy and F1-score in most cases, particularly in cases where the dataset is highly diverse and imbalanced, highlighting the robustness of the contrastive learning approach.

II. OUR CONTRASTIVE LEARNING FRAMEWORK

Contrastive learning is leveraged to align bug reports of similar severity in the embedding space. Specifically, we use NT-Xent (Normalized Temperature-scaled Cross Entropy) loss [8] to encourage semantically similar reports to cluster together, while pushing apart dissimilar reports. Our framework (see Fig. 1) consists of three steps:

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), grant 2018-06588.

¹https://github.com/seer-lab/bug_severity_contrastive_learning

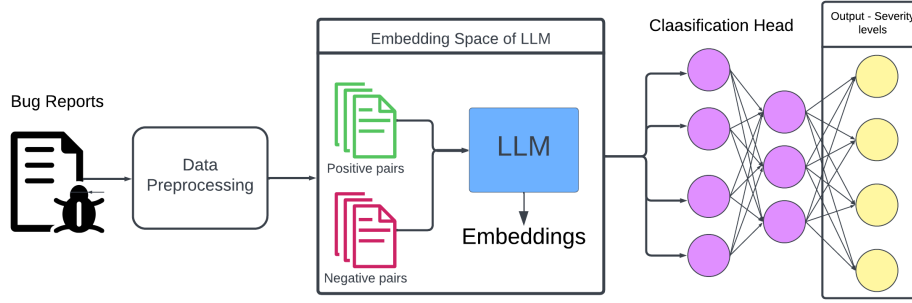


Fig. 1: Overview of Bug Severity Classification Approach

(1) **Pre-processing and pairing bug reports.** Positive pairs are formed from bug reports sharing the same severity label; negative pairs are drawn from reports with differing severity levels. No textual augmentation (e.g., paraphrasing, masking) is used and similarity is defined solely based on class label.

(2) **Generating embeddings using a contrastive learning strategy applied to a LLM.** Each report in a pair is passed through CodeBERT [9] to generate embeddings. CodeBERT is an encoder-only transformer model containing code-comment pairs across multiple programming languages. It is well-suited to the task of bug severity prediction as bug reports often include technical terminology and code snippets. The bug descriptions in each report are tokenized and truncated or padded to a maximum length of 256 tokens. The tokenized bug descriptions and input IDs with attention masks are what is passed into CodeBERT². The model outputs a 768-dimensional embedding representing the bug report’s semantics. Once the embeddings for each pair of reports are generated, pairwise cosine similarity [10] is computed between the embeddings. NT-Xent loss is applied to pull positive pairs closer and push negative pairs apart in the embedding space.

(3) **Training a classification head on the learned embeddings to predict severity.** After contrastive training, a linear classification head is trained using the embeddings to predict the severity class. Cross-entropy loss [11] is used for classification.

III. DATASET

Our research focuses on the use of two main datasets – the NASA Project and Issue Tracking System (PITS) dataset [12] and the Mozilla Bug Reports dataset [13]. We selected these two datasets because they are distinctly different examples from the literature.

On one hand, the NASA PITS dataset is from a closed-source development environment and is small in size. It is made up of five NASA robotic mission projects: PITS A (965 reports), PITS B (987 reports), PITS C (323 reports), PITS D (182 reports), and PITS E (825 reports) with defined severity

²The hyperparameters for CodeBERT fine-tuning (learning rate = 2e-5, batch size = 32, temperature = 0.5) were selected based on commonly adopted values in LLM literature and preliminary tuning on validation data. Due to computational constraints, a full grid search was not conducted.

TABLE I: Model Architecture and Training Configuration

Component	Details
Backbone	CodeBERT (RoBERTa-based pre-trained model)
Tokenizer	RoBERTaTokenizer (HuggingFace)
Input Length	256 tokens
Embedding Dimension	768
Contrastive Loss	NT-Xent Loss (Temperature = 0.5)
Similarity Metric	Cosine Similarity
Positive Pairing	Reports with same severity
Negative Pairing	Reports with different severities
Text Augmentation	None (pairing based only on labels)
Classification Head	Fully connected layer (768 → num_classes)
Loss Function	Cross-Entropy Loss
Optimizer	Adam
Learning Rate	2e-5
Batch Size	32

levels (2-5). The dataset is imbalanced in severity levels, with most bugs falling under levels 2 and 3, while levels 4 and 5 are less common. A larger dataset is produced by combining all the projects and is referred as Combined PITS (3,282 bug reports). Each bug report in the dataset includes a Project ID, creation date, subject, severity, and description. For this study, only the description and severity are utilized.

On the other hand, the Mozilla Bug Reports dataset is from open-source development and is larger with 9,998 bug reports across five severity levels (1, 2, 4, 5, and 6). The majority of the bugs belong to severity 2, while severity 6 is the least populated. Bug reports in the dataset include a Bug ID, Creation date, Component Name, Product Name, status, short description, long description, severity levels, etc. For this study, only long description and the corresponding severity levels are utilized. However, when evaluating against unseen projects, product names are also considered to enhance the cross-project generalization.

The severity levels differ slightly between the PITS and Mozilla datasets and there is no standardized mapping between the two. Furthermore, the criteria for assigning severity may vary across projects. As a result we focus on using each dataset independently for intra-dataset and cross-project experiments.

IV. EXPERIMENTAL DESIGN

Our experimental evaluation focused on answering the following questions to gain insight into the use of contrastive

learning and LLMs in bug severity classification:

- **RQ1:** How effective is the proposed contrastive learning model in classifying bug severity?
 - **RQ1(a):** When applied within the same project?
 - **RQ1(b):** When trained on past projects and tested on a new or unseen project?
- **RQ2:** How does training with contrastive learning improve bug severity classification compared to non-contrastive approaches?
- **RQ3:** How well do LLMs classify bug severity across applications compared to traditional word embedding methods like Doc2Vec?

A series of experiments were conducted to address each research question and evaluate the effectiveness of the proposed contrastive learning-based approach for bug severity classification. Two datasets are used in this experiment - the PITS dataset and the Mozilla dataset. Three models are evaluated - the proposed approach of using contrastive learning to fine-tune LLM, fine-tuning LLM without contrastive learning, and baseline model Doc2vec+MLP from the research [14].

Experiment 1: Train and Test on the Same Project: The goal of this experiment is to evaluate the performance of the models when trained and tested on the same project datasets, addressing **RQ1(a)**. The evaluation is conducted on individual PITS datasets and a combined dataset that aggregates all the PITS projects together. Additionally, the models are evaluated on the larger Mozilla dataset. For each dataset, 80% of the data is used for training and 20% for testing. This experiment provides insights into the model’s ability to accurately classify bug severity within the same project environment and highlights its performance across datasets of varying sizes and distributions.

Experiment 2: Train on One or More Projects and Test on an Unseen Project: The objective of this experiment is to evaluate the generalizability of the models when trained on one or more project datasets and tested on an unseen project, addressing **RQ1(b)**. The models are trained on various combinations of PITS and Mozilla datasets and tested on a different, unseen PITS and Mozilla project (see Table III for combination details). This experiment assesses how well the models adapt to unseen project data and measures their robustness in cross-project scenarios. Due to the large class mismatch and differing label bug severity sets, direct cross-dataset evaluation was avoided. Future work could consider harmonized label mappings to bridge such domain gaps.

Experiment 3: Contrastive Learning vs. Non-Contrastive Learning: The goal of this experiment is to compare the performance of the LLM model fine-tuned using our contrastive learning-based framework with the LLM model trained traditionally, using a non-contrastive learning approach. In the non-contrastive approach, the model is directly fine-tuned for classification. This experiment addresses **RQ2**. The comparison is conducted on the PITS datasets and the Mozilla dataset, following the settings of *Experiment 1* and *Experiment 2*. Models are evaluated using accuracy and F1-score to

determine the benefits of contrastive learning for bug severity classification.

Experiment 4: LLMs vs. Traditional Embedding Methods: This experiment compares the performance of LLMs, specifically CodeBERT, with traditional embedding methods like Doc2Vec [14]. The experiment addresses **RQ3**. The evaluation is performed on the PITS datasets (A, B, C, D, E, and Combined PITS) and the Mozilla dataset, as described in *Experiment 1* and *Experiment 2*. For LLMs, embeddings are extracted using CodeBERT, fine-tuned with and without contrastive learning.

The baseline approach from [14] is replicated, where Doc2Vec embeddings are used as input to an Multi-Layer Perceptron (MLP) classifier. In *Experiment 1* (same-project settings), Doc2Vec is trained separately on both descriptions and summaries to provide a comprehensive comparison. However, in *Experiment 2* (cross-project settings), the Doc2Vec model trained on descriptions struggled to classify bug reports effectively. As a result, Doc2Vec model is only trained on summaries and evaluated against LLM-based models, which use report descriptions to classify bug severity. All other experimental settings from [14] are maintained for consistency.

Overfitting Mitigation: To prevent overfitting on smaller datasets, the model’s performance is continuously monitored by evaluating the validation loss during training. Regularization techniques, such as L1 and L2 penalties [15], are applied to reduce the complexity of the model and improve its generalization capability. Additionally, an early stopping mechanism is employed, halting training when the validation loss stops improving, further minimizing the risk of overfitting.

V. RESULTS

Performance Comparison

The results presented in this section focus on the accuracy and F1-score for each model configuration and dataset. They comprehensively compare contrastive learning-based approaches, non-contrastive models, and traditional embedding methods (e.g., Doc2Vec). We use accuracy because it provides a general measure of the model’s performance although it may not account for class imbalances in the dataset. We use the F1-score because it is particularly useful for datasets with imbalanced severity levels, as it highlights the trade-off between precision and recall.

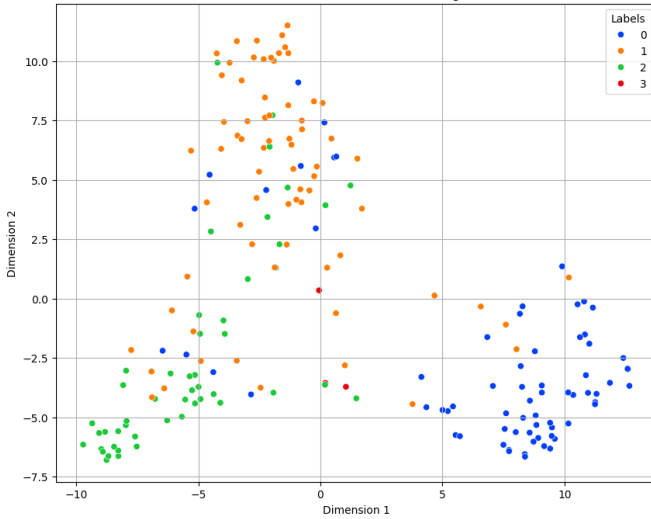
Table II demonstrates the performance of the models on the same project dataset with setups from *Experiment 1*, addressing **RQ1(a)**, **RQ2**, and **RQ3**. The results demonstrate that contrastive learning consistently outperforms other approaches across all datasets, including Combined PITS, individual PITS (A, B, C, D, E), and Mozilla. For example, Contrastive Learning achieved an accuracy of 0.7686 and F1-Score of 0.7615 on the Combined PITS dataset, outperforming the non-contrastive approach, which scored 0.7458 and 0.7304, respectively. Traditional methods like Doc2Vec+MLP performed comparatively worse, although using summaries with Doc2Vec+MLP provided better results, particularly on Mozilla, which achieved the highest accuracy of 0.8235.

TABLE II: Performance Comparison of Models on seen Projects

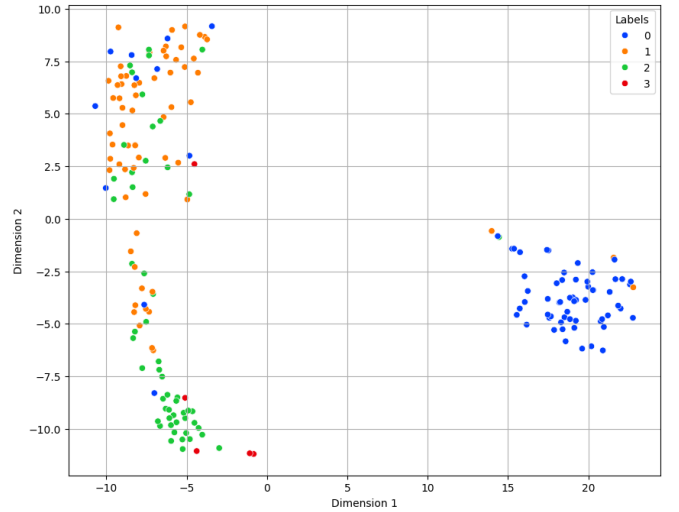
Datasets	Contrastive Learning		Without Contrastive Learning		Doc2vec+MLP with Description [14]		Doc2vec+MLP with Summary [14]	
	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1
Combined PITS	0.77	0.76	0.75	0.73	0.65	0.65	0.73	0.72
PITS A	0.77	0.77	0.74	0.73	0.66	0.66	0.75	0.75
PITS B	0.72	0.72	0.69	0.68	0.67	0.69	0.64	0.64
PITS C	0.91	0.90	0.88	0.88	0.86	0.85	0.89	0.89
PITS D	0.97	0.98	0.97	0.97	0.97	0.98	0.96	0.96
PITS E	0.76	0.75	0.65	0.66	0.62	0.63	0.72	0.71
Mozilla	0.81	0.78	0.78	0.77	0.74	0.75	0.82	0.76

TABLE III: Performance Comparison of Models on Unseen Projects

Datasets (PITS)	Contrastive Learning		Without Contrastive Learning		Doc2vec+MLP with Summary [14]	
Train-Test	Accuracy	F1	Accuracy	F1	Accuracy	F1
A, B, C, D - E	0.56	0.55	0.50	0.50	0.61	0.51
B, C, E - D	0.49	0.59	0.30	0.38	0.44	0.56
A, B - C	0.68	0.67	0.67	0.67	0.80	0.82
C, E - B	0.61	0.58	0.56	0.52	0.51	0.51
Mozilla (Core)	0.78	0.75	0.79	0.71	0.64	0.68
Mozilla (Firefox)	0.80	0.77	0.79	0.76	0.67	0.69



(a) LLM embedding without contrastive learning



(b) LLM embedding with contrastive learning

Fig. 2: Comparison of LLM embedding visualizations: (a) without contrastive learning, (b) with contrastive learning.

On simpler datasets such as PITS D, all methods performed similarly, highlighting the minimal complexity of the dataset. However, for larger and more diverse datasets like Combined PITS and Mozilla, Contrastive Learning showcased its ability to generalize better across varied bug reports, achieving the highest F1-Scores. These findings emphasize the strength of fine-tuned LLMs, particularly with contrastive learning, in handling complex bug severity classification tasks, outperforming traditional embedding methods like Doc2Vec+MLP.

A comparative analysis of the performance of the evaluated models on seen projects (see Table II), further reinforces the effectiveness of contrastive learning. The trend shows that the contrastive learning approach has consistently higher F1 scores

across all datasets. F1 scores are demonstrated since it is a better measure for imbalanced datasets.

The results from models trained on one or more projects and tested on an unseen project (see Table III), address **RQ1(b)**, **RQ2**, and **RQ3**. We report LLM-based results for descriptions and Doc2Vec+MLP [14] results for summaries due to a limitation with Doc2Vec+MLP where it failed to classify bug reports from descriptions (accuracy below 0.50). Therefore, the results highlight a performance gap between traditional embedding models and LLMs on unseen projects.

Contrastive learning consistently outperforms non-contrastive in most cases. For example, when tested on PITS E (trained on A, B, C, and D), Contrastive Learning

achieves 0.5636 accuracy and 0.549 F1-Score, compared to 0.497 and 0.5047 for non-contrastive learning. However, Doc2Vec+MLP [14] performs best in some cases, such as PITS E, with 0.6073 accuracy. In challenging cases like PITS D (trained on B, C, E), Contrastive Learning achieves an F1-Score of 0.5922, outperforming non-contrastive learning (0.381) and Doc2Vec+MLP (0.5646). This highlights the robustness of contrastive learning on imbalanced, diverse datasets, where non-contrastive methods fail. For PITS C (trained on A and B), Doc2Vec+MLP achieves the highest accuracy (0.7988) and F1-Score (0.8249), leveraging summaries effectively for the smaller dataset. However, Contrastive Learning performs best for PITS B (trained on C and E) due to its ability to learn robust patterns from more diverse and highly imbalanced datasets, resulting in better generalization.

In the Mozilla dataset, Contrastive Learning also demonstrates strong performance. When trained on the entire Mozilla dataset excluding Core-related bug reports and tested on the Core subset, it achieves an accuracy of 0.78 and an F1-Score of 0.75, outperforming non-contrastive learning (0.79 accuracy, 0.71 F1) and Doc2Vec+MLP (0.64 accuracy, 0.68 F1). Similarly, when trained on the entire Mozilla dataset excluding Firefox-related bug reports and tested on the Firefox subset, Contrastive Learning achieves 0.80 accuracy and 0.77 F1-Score, while non-contrastive learning scores 0.79 and 0.76, and Doc2Vec+MLP lags behind at 0.67 and 0.69. These results reinforce the effectiveness of contrastive learning in handling severity classification in diverse datasets like Mozilla.

When we consider the model performance on unseen projects (see Table III), the results highlight that contrastive learning improves cross-project generalization, outperforming non-contrastive models in most cases, though traditional methods like Doc2Vec+MLP show higher accuracy in specific cases (e.g., PITS C). F1 scores are preferred for comparison, since it is a better measure for imbalanced datasets.

Explainability of the Proposed Model

To understand the impact of contrastive learning, t-Distributed Stochastic Neighbor Embedding (t-SNE) visualizations are generated to illustrate the distribution of bug report embeddings in a two-dimensional space. In Figure 2a, we visualize the embeddings without contrastive learning. We can observe that severity classes (0, 1, 2, and 3) overlap, indicating that the model struggles to produce distinct representations for each class. This overlap suggests a challenge in accurately classifying bug severities, particularly for ambiguous or underrepresented classes. In Figure 2b, we visualize the embeddings with contrastive learning and observe improved separation between severity classes. Each class forms more distinct clusters with reduced overlap. Overall, the t-SNE plots demonstrate that contrastive learning enhances the model’s ability to generate discriminative embeddings, which improves classification performance, as observed in the accuracy and F1-Score results.

VI. DISCUSSION

The proposed contrastive learning-based approach significantly improves bug severity classification compared to non-contrastive methods and traditional embeddings like Doc2Vec [14]. Fine-tuning CodeBERT with contrastive learning produces more distinguishable embeddings, as seen in t-SNE visualizations, leading to higher accuracy and F1-scores, especially on larger datasets with richer descriptions.

LLMs (e.g., CodeBERT) outperform Doc2Vec+MLP on larger, diverse datasets, excelling in handling unstructured descriptions where contextual understanding is critical. Doc2Vec+MLP performs competitively on smaller datasets or structured summaries, requiring fewer computational resources. However, it struggles with report descriptions, showing accuracies below 0.50 in cross-project evaluations (Experiment 2), highlighting its limitations with long, unstructured text.

It was observed that input length did not significantly affect the performance of CodeBERT, as consistent classification accuracy was maintained across both long descriptions and short summaries. In contrast, Doc2Vec+MLP showed better performance on summaries than full descriptions, particularly in smaller datasets. These findings suggest that CodeBERT is more robust in handling long report description due to its contextual encoding capabilities.

The performance decline in *Experiment 2* reflects challenges with imbalanced severity distributions and domain shifts across projects. Overrepresented classes (e.g., labels ‘2’ or ‘3’) dominate, making underrepresented classes harder to predict. Contrastive learning improves LLM robustness by aligning embeddings for minority classes. In summary, LLMs with contrastive learning excel on complex datasets, while Doc2Vec+MLP is effective for smaller, structured inputs but struggles with long and detailed bug descriptions.

Threats to Validity

Although this study shows promising results for automatic bug severity classification, there are several threats to the generalizability of the results. First, the results are based on two datasets (PITS, Mozilla) and a single LLM for generating embeddings (CodeBERT). Experiments with more diverse data and LLMs is needed to fully understand the benefits of contrastive learning in bug severity classification. Second, over-representation of certain severity levels hinders learning for underrepresented classes. It is not clear how the models may perform with a more balanced dataset.

While we did not conduct formal statistical significance tests, our evaluation included diverse datasets, multiple train-test configurations, and uniform experimental conditions. The consistent improvements in both F1-score and accuracy across these varied settings suggest that the performance gains from contrastive learning are reliable. Nevertheless, we acknowledge the need for repeated trials with randomized seeds and statistical testing (e.g., paired t-tests or bootstrap confidence intervals) in future work to rigorously confirm the significance of these findings.

VII. RELATED WORKS

Previous approaches to automatic bug severity classification include traditional machine learning, deep learning, and transformer-based models. Ahmed et al. [16] proposed CaPBug, a framework that utilizes TF-IDF feature extraction along with classifiers such as Naïve Bayes and Random Forest, and incorporates SMOTE (Synthetic Minority Over-sampling Technique) to handle imbalanced class distributions. Agrawal and Goyal [17] applied Word2Vec embeddings, showing improved accuracy over keyword-based methods, but lacked contextual adaptability. Deep learning approaches like BiLSTM combined with GloVe embeddings achieved high performance on benchmark datasets, with precision exceeding 90% [18]. Kim and Yang [19] proposed a CNN-LSTM hybrid, outperforming earlier baselines through topic-based feature selection. More recently, transformer-based models have shown promise. Ali et al. [20] fine-tuned BERT on mobile app bug reports, achieving significant accuracy gains. Jiang et al. [21] presented NOVA, which utilizes contrastive learning and hierarchical attention for binary code tasks, demonstrating the effectiveness of structured representation learning. Our work builds on these foundations by applying contrastive learning to LLM embeddings for bug severity classification. This enhances generalization, especially on imbalanced or cross-project data, and outperforms both traditional Machine Learning and fine-tuned LLM baselines.

VIII. CONCLUSION

Our novel framework integrates LLM embeddings with contrastive learning to automatically predict bug severity. It can benefit software maintenance by automatically recommending or assigning a severity level based on a bug report's text, thus assisting developers and triage teams in prioritizing issues more consistently and efficiently. The experimental evaluation with the NASA PITS dataset and the Mozilla dataset demonstrates *fine-tuning LLMs with contrastive learning consistently improves performance, particularly on imbalanced and diverse datasets, compared to non-contrastive methods*. Furthermore, the findings highlight the effectiveness of LLMs in handling longer bug report descriptions, outperforming traditional embedding models like Doc2Vec+MLP in such scenarios.

In the future, we plan to expand our evaluation to a more diverse set of projects to better understand the generalizability and robustness of the proposed framework. We are also interested in conducting structured error analysis and including qualitative examples of success/failure cases to provide more insight into the quality of the bug severity prediction.

Another future direction is to evaluate our framework with other language models. On the one hand we would like to consider lightweight models (e.g., TinyBERT [22]) and on the other hand, we aim to explore newer LLMs (e.g., CodeT5+ [23]) to assess their suitability. Evaluating different LLMs would provide insight into the trade-offs between computational efficiency and performance.

REFERENCES

- [1] L. A. F. Gomes, R. da Silva Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Information and software technology*, vol. 115, pp. 58–78, 2019.
- [2] S. G. Jindal and A. Kaur, "Automatic keyword and sentence-based text summarization for software bug reports," *IEEE Access*, vol. 8, pp. 65 352–65 370, 2020.
- [3] J. Kim and G. Yang, "Bug severity prediction algorithm using topic-based feature selection and CNN-LSTM algorithm," *IEEE Access*, vol. 10, pp. 94 643–94 651, 2022.
- [4] R. Hazra, A. Dwivedi, and A. Mukherjee, "Is this bug severe? a text-cum-graph based model for bug severity prediction," in *Proc. of. Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2022, pp. 236–252.
- [5] L. Kumar, M. Kumar, L. B. Murthy et al., "An empirical study on application of word embedding techniques for prediction of software defect severity level," in *Proc. of. the 16th Conference on Computer Science and Intelligence Systems (FedCSIS 2021)*, 2021, pp. 477–484.
- [6] L. Ning, L. Liu, J. Wu et al., "User-LLM: Efficient LLM contextualization with user embeddings," *arXiv preprint arXiv:2402.13598*, 2024.
- [7] H. Hu, X. Wang, Y. Zhang, Q. Chen, and Q. Guan, "A comprehensive survey on contrastive learning," *Neurocomputing*, p. 128645, 2024.
- [8] P. Khosla, P. Teterwak, C. Wang et al., "Supervised contrastive learning," *Advances in neural information processing systems*, vol. 33, pp. 18 661–18 673, 2020.
- [9] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.
- [10] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," in *Proc. of the 4th International Conference on Cyber and IT Service Management*, 2016, pp. 1–6.
- [11] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.
- [12] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proc. of the IEEE International Conference on Software Maintenance*, 2008, pp. 346–355.
- [13] N. K.-S. Roy and B. Rossi, "Towards an improvement of bug severity classification," in *Proc. of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2014, pp. 269–276.
- [14] J. Arokiam and J. S. Bradbury, "Automatically predicting bug severity early in the development process," in *Proc. of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 17–20.
- [15] I. Nusrat and S.-B. Jang, "A comparison of regularization techniques in deep neural networks," *Symmetry*, vol. 10, no. 11, p. 648, 2018.
- [16] H. A. Ahmed, N. Z. Bawany, and J. A. Shamsi, "CaPBug-A framework for automatic bug categorization and prioritization using NLP and machine learning algorithms," *IEEE Access*, vol. 9, pp. 50 496–50 512, 2021.
- [17] R. Agrawal and R. Goyal, "Developing bug severity prediction models using word2vec," *International Journal of Cognitive Computing in Engineering*, vol. 2, pp. 104–115, 2021.
- [18] T. S. Mian and A. Alsaedi, "Software bug severity prediction using convolutional neural network and bilstm models," in *International Conference of Reliable Information and Communication Technology*. Springer, 2023, pp. 1–12.
- [19] J. Kim and G. Yang, "Bug severity prediction algorithm using topic-based feature selection and CNN-LSTM algorithm," *IEEE Access*, vol. 10, pp. 94 643–94 651, 2022.
- [20] A. 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, 2024.
- [21] N. Jiang et al., "Nova: Generative language models for assembly code with hierarchical attention and contrastive learning," in *Proc. of the 13th International Conference on Learning Representations*, 2025.
- [22] X. Jiao, Y. Yin, L. Shang et al., "TinyBERT: Distilling BERT for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.
- [23] Y. Wang, H. Le, A. Gotmare et al., "CodeT5+: Open code large language models for code understanding and generation," in *Proc. of the Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 1069–1088.