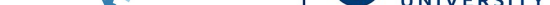


Software Engineering & Education Research Lab
Ontario Tech University, Oshawa, Canada



SEERLAB | OntarioTech UNIVERSITY



Why Bug Severity Classification still needs work?

- Misclassification of bugs delays fixes and increases costs
- Manual labelling is subjective and inconsistent
- Traditional Machine Learning models relied on critical-term heuristics and lacked adaptability [1]
- Deep models (CNN-LSTM) improved accuracy but lacked generalization [2]
- Contextual models (ELMo, BERT) struggled on imbalanced data [3]



Improve robustness and generalization across projects

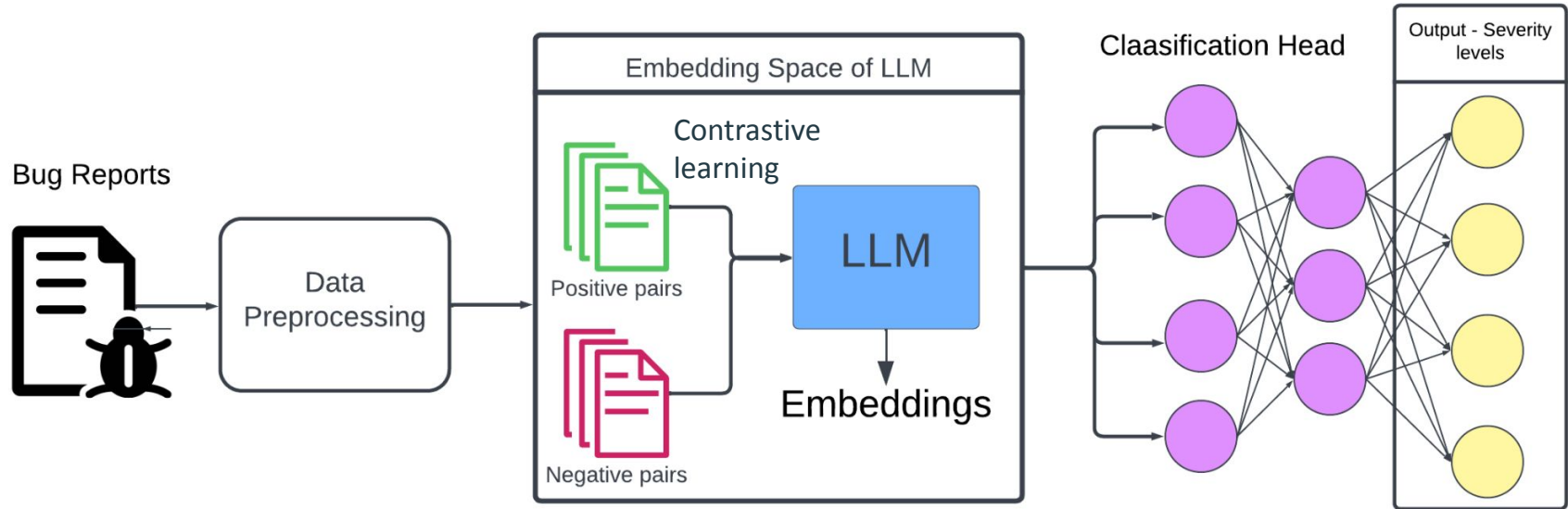


CodeBERT + Contrastive Learning for better severity separation

Key Contributions

- **Contrastive fine-tuning for LLMs** improves semantic separation and boosts generalization on imbalanced, diverse datasets
- **Comprehensive evaluation** compares our model against baseline approaches across NASA and Mozilla datasets
- Demonstrates strong **cross-project robustness** on unseen project data
- Offers **empirical insights** on the effects of **class imbalance** and **dataset size**
- Presents a **scalable, automated solution** for real-world bug triage systems

Overview of proposed approach



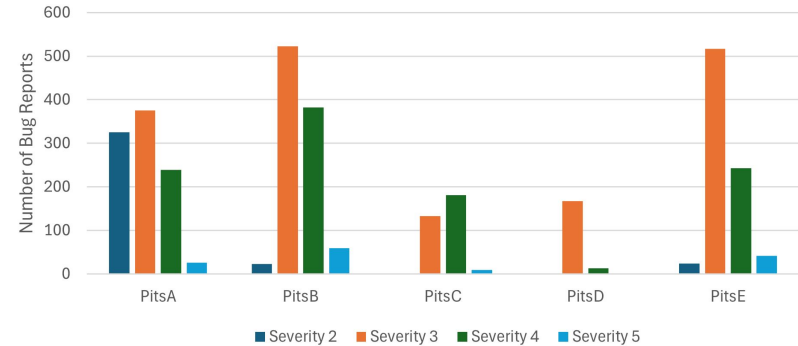
Construct:

- **Positive Pairs** - Two bug reports with **same** severity levels
- **Negative pairs** - Two bug reports with **different** severity levels.
- Generate embeddings using CodeBERT with contrastive loss (NT-Xent)
- Generate an embedding space with positive pairs closer and negative pairs far away from each other.
- Train classifier on learned embeddings using cross-entropy loss.

Datasets

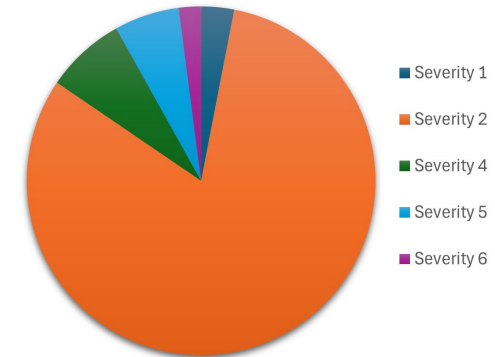
NASA PITS Dataset

- 5 closed-source NASA projects: PITS A – E (total: 3,282 reports)
- Severity levels: 2–5 (imbalanced, most reports in 2 & 3)
- Used bug **description** and **severity label**



Mozilla Bug Reports

- Open-source: 9,998 reports across severities 1, 2, 4, 5, 6
- Richer and longer descriptions; more diversity
- Used **descriptions** and **severity levels** used
- **Product name** used in cross-project experiments to boost generalization



Research question

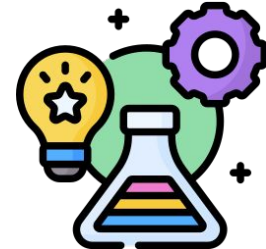


RQ1: How effective is contrastive learning based LLM for bug severity classification-

- (a) Within the **same project**?
- (b) On **unseen projects**?

RQ2: Does **contrastive learning** based fine tuning improve the performance of LLMs over **non-contrastive** based fine-tuning?

RQ3: How do **LLMs** compare to **traditional methods** models like Doc2Vec?



Experiments

1. Experiment 1 – Train/test on same project using an 80/20 split → **RQ1(a)**
2. Experiment 2 – Train on some projects, test on unseen project → **RQ1(b)**
3. Experiment 3 – Contrastive vs. non-contrastive on exp 1 and 2 settings → **RQ2**
4. Experiment 4 – CodeBERT vs. Doc2Vec+MLP [4] n exp 1 and 2 settings → **RQ3**

Table 1: Dataset setup for
Experiment 1

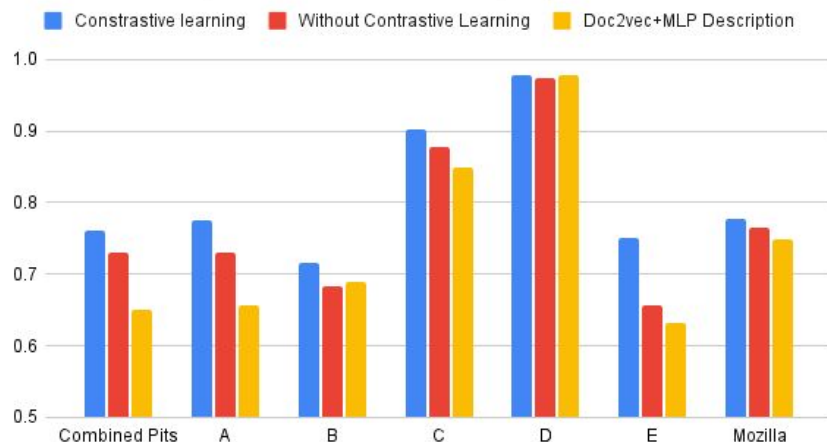
Train	Test
Combined Pits (80%)	20%
Pit A (80%)	20%
Pit B (80%)	20%
Pit C (80%)	20%
Pit D (80%)	20%
Pit E (80%)	20%
Mozilla (80%)	20%

Table 2: Dataset setup for
Experiment 2

Train	Test
Pits A, B, C, D	Pit E
Pits B, C, E	Pit D
Pits A, B	Pit C
Pits C, E	Pit B
Mozilla except Core	Core
Mozilla except FIREFOX	FIREFOX

Results

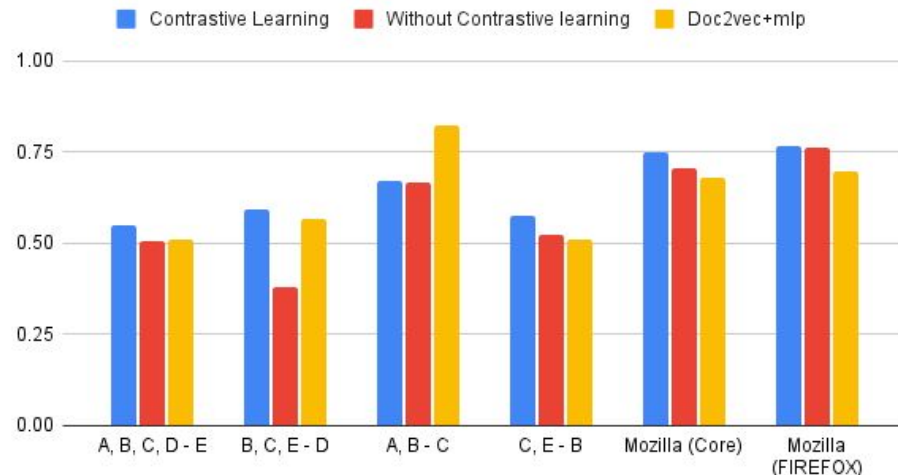
F1 Score Comparison across same project



F1 Score – Same Project

- **Contrastive Learning** consistently achieved the **highest F1-scores**
- Strongest gains on **Combined PITS** and **Mozilla** (diverse, imbalanced datasets)

F1 Score comparison on unseen project



F1 Score – Unseen Project

- **Contrastive Learning** generalizes better across unseen projects
- Exception: **PITS C**, where **Doc2Vec+MLP** performed best using summaries

Explainability of the proposed model

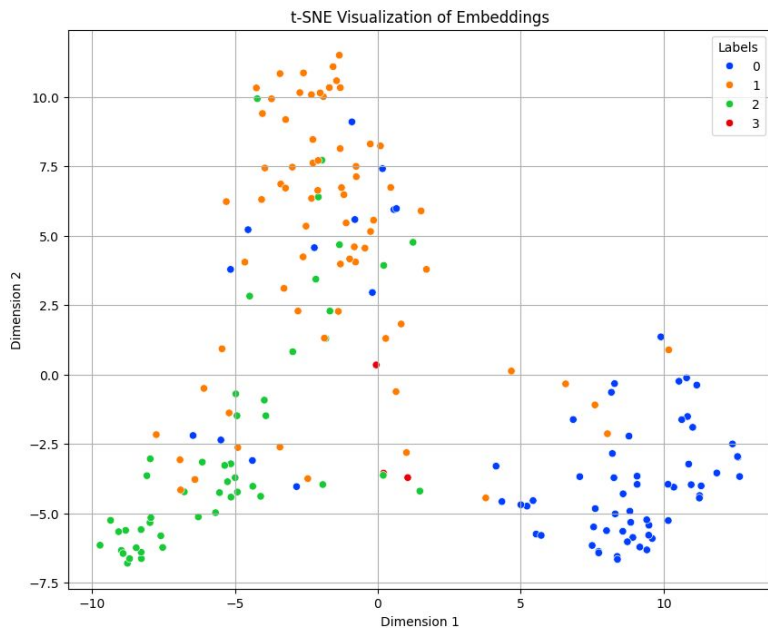


Fig 1: LLM Embedding visualization **without** contrastive learning

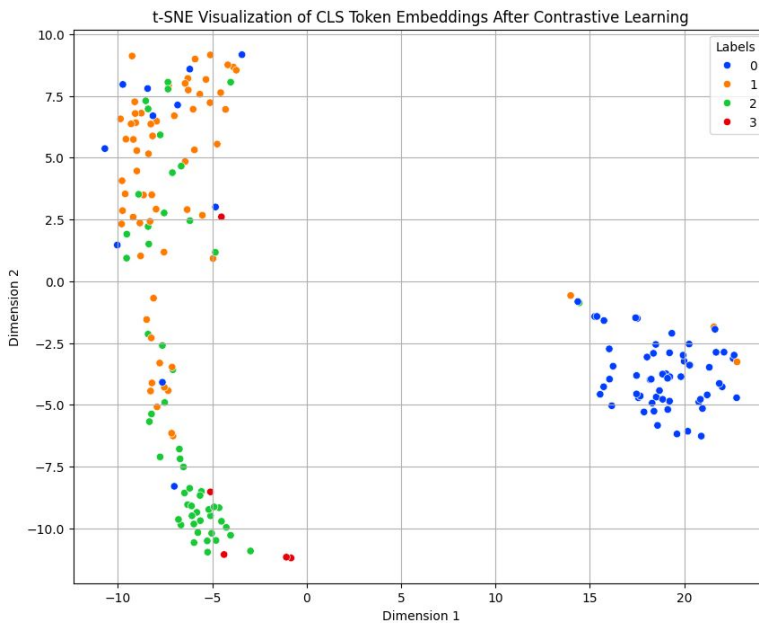


Fig 2: LLM Embedding visualization **with** contrastive learning

Key Insights

1. **Contrastive Learning = Better Class Separation**
 - Improves representation for minority classes
 - Boosts robustness in cross-project settings
2. **LLMs vs Traditional Methods:**
 - CodeBERT excels on longer, unstructured reports
 - Doc2Vec works better on summaries in small datasets
3. **Generalization Challenge:**
 - Unseen projects cause drop in performance
 - Contrastive learning narrows this gap
4. **Input Length Robustness:**
 - CodeBERT unaffected by length variations
 - Doc2Vec struggles with full descriptions



Limitation and future direction

Limitations

- Evaluated on only two datasets (NASA PITS, Mozilla) using a single LLM (CodeBERT)
- Not clear how the models would work on balanced datasets
- Did not perform any statistical tests by repeating experiments.

Future Work:

- Extend evaluation:
 - more datasets from diverse domains
 - more LLMs (e.g., TinyBERT, CodeT5+)
 - apply statistical tests (e.g., t-tests, confidence intervals)
- Align severity labels across datasets for cross-domain generalization

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

Mosarrat Rumman - mosarrat.rumman@ontariotechu.net

Emon Roy - emon.roy@ontariotechu.net

Anushka Zaman - anushka.zaman@ontariotechu.net

Jeremy S. Bradbury - jeremy.bradbury@ontariotechu.net

In collaboration with:

**Software Engineering & Education
Research Lab (SEER LAB)**

Ontario Tech University, Oshawa, Canada

<http://www.seerlab.ca>

Reference

1. Sharma, G., Sharma, S., & Gujral, S. (2015, November 21). *A novel way of assessing software bug severity using dictionary of critical terms*. Procedia Computer Science. Retrieved from <https://www.sciencedirect.com/science/article/pii/S1877050915032238>.
2. Kim, J., & Yang, G. (n.d.). Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm. Retrieved from <https://www.iitrpr.ac.in/library/pdf/IEEE.pdf>.
3. Chen, B., Zou, W., Cai, B., Meng, Q., Liu, W., Li, P., & Chen, L. (2024, July 25). *An empirical study on the potential of word embedding techniques in bug report management tasks - empirical software engineering*. Retrieved from. <https://link.springer.com/article/10.1007/s10664-024-10510-3>.
4. Arokiam, J., & Bradbury, J. S. (2020, June). Automatically predicting bug severity early in the development process. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results (pp. 17-20).

