# DEFTri: A Few-Shot Label Fused Contextual Representation Learning For Product Defect Triage in e-Commerce

**Ipsita Mohanty**
Walmart Global Tech
Sunnyvale, California, USA
ipsita.mohanty@walmart.com

## Abstract

Defect Triage is a time-sensitive and critical process in a large-scale agile software development lifecycle for e-commerce. Inefficiencies arising from human and process dependencies in this domain have motivated research in automated approaches using machine learning to accurately assign defects to qualified teams. This work proposes a novel framework for automated defect triage (DEFTri) using fine-tuned state-of-the-art pre-trained BERT on labels fused text embeddings to improve contextual representations from human-generated product defects. For our multi-label text classification defect triage task, we also introduce a Walmart proprietary dataset of product defects using weak supervision and adversarial learning, in a few-shot setting.

## 1 Introduction

In large e-commerce organizations, there are many defects generated periodically with a massive pool of software teams and developers spread across geographies to pick from, each with unique domain specialization. Most organizations have a large pool of human triaging agents responsible for routing these product defects across various teams within the organization. However, large-scale software releases are time-sensitive, and effective defect assignments are a critical component in the process that is prone to bottlenecks. Determining the most suitable team to own a defect may require several attempts; thus, wasting time to diagnose a defect not in the team's domain of specialty and, overall, negatively impacting the defect resolution throughput.

Prior industry research work on automated defect triage has primarily focused on using the traditional machine learning approaches. However, with the recent surge of state-of-the-art pre-trained language models, one under-explored field of application is operations in agile software development. In the defect triage, handling scenarios require Natural Language Understanding to utilize the context of the defects logged by human testers, to predict all the teams associated with resolution. The current defect triage process is primarily human-agents driven. This work integrates an automated defect triage framework, DEFTri using product defect's contextual features to achieve operational excellence within Walmart's software development lifecycle.

We propose a novel framework, DEFTri to perform an automated defect triage using contextual representations of human-generated defect texts. We use Walmart's proprietary data of product defects curated by product managers, program managers, and beta-testers to train our models. We use domain-specific lexicons to generate labeled training data using weak supervision in a few shot settings. We further use adversarial learning to increase our training sample size while increasing the robustness of our models. We propose our model architecture for fine-tuning pre-trained BERT (Devlin et al., 2018) for our multi-label classification task. Finally, we consolidate our experiments, analyze the results and discuss future research work.

## 2 Related Work

Prior research work on defect triage (Choquette-Choo et al., 2019; Mani et al., 2018; Soleimani Neysiani et al., 2020) mostly focuses on using traditional machine learning and RNNs on word vector representations of text using BOW, Word2Vec, Tfidf, etc. Another recent research relies on graph representation learning for defect triage (Wu et al., 2021). This paper proposes a graph recurrent convolution network with a joint random walk mechanism-based architecture. Also, several recent research on label embedding (Xiong et al., 2021; Liu et al., 2021; Si et al., 2020) has shown promising results for learning the text and label representation in the same latent



*Proceedings of The Fifth Workshop on e-Commerce and NLP (ECNLP 5)*, pages 1 - 7

space. We further the research by proposing a novel architecture to derive superior contextual text representations using state-of-the-art language model BERT for multi-label defect triage.

Most of these published research benchmarks are on open-source defect report datasets - Eclipse and Mozilla (Lamkanfi et al., 2013). However, these datasets are focused on technical errors generated during system failures and do not mimic our use case. Our product defects are comprehensive user testing reviews consisting of natural language, technical and domain-specific text. In the real world, gathering labeled data is hard and expensive. Hence, we propose a methodology to generate a robust proprietary multi-label training dataset using weak supervision and adversarial learning.

## 3 Data

Our primary dataset is a proprietary in-house dataset consisting of actual defect reviews generated by beta testers for one of our major software releases. We rely on defect title and description fields to create the text corpus and text labels to identify the teams uniquely. Each defect could have multiple associated teams and vice versa. For our research, we have 3485 samples as a train set and 85 samples as the test set with 15 unique team labels for our multi-label dataset. Refer Table 1. We have 4-5 human-expert annotated defects corresponding to each team label in our low-resource setting. Our data preparation pipeline follows the below steps,

### 3.1 Generate Labeled Data Using Weak Supervision

Despite the success of fine-tuning pre-trained language models, one bottleneck is the requirement of labeled data. These labeled training data were expensive and time-consuming to create. It required human annotators with domain expertise to read through each defect review and assign team labels accordingly. Every change in labeling guidelines, team orientation, or use case changes necessitated re-labeling. Hence, we used Snorkel label model (Ratner et al., 2017) to generate weak labels for our training data. We apply 25 labeling functions (LFs) to unlabelled training data using a snorkel pipeline. Refer Table 2.

### 3.2 Generate Synthetic Data Using Adversarial Learning

Machine learning algorithms are often vulnerable to adversarial examples that have imperceptible alterations from the original counterparts but can fool the state-of-the-art models (Jin et al., 2019; Dong et al., 2021).To increase the robustness, model training can be done using adversarial examples (Goodfellow et al., 2014; Gowal et al., 2021). We use Textattack framework (Morris et al., 2020) on 30% of our data, chosen at random to generate synthetic data for training our models and append these synthetic examples to our train set. We use embedding recipe of the framework that augments text by replacing words with neighbors in the counter-fitted embedding space, with a constraint to ensure their cosine similarity is at least 0.8. For every sampled defect, we produce 2 augmented defect texts by altering 10% of original text words, while preserving the team labels . Refer Table 3

### 3.3 Fix Data Imbalances

We found that the final training data created using the above techniques were imbalanced. This issue was because the product defects were likely skewed towards a specific defect associated with a more significant and frequently tested domain vs. a rarely occurring one. We also noticed that defect reviews for features related to new team labels are getting introduced into the environment on an ongoing basis. To resolve the skewness, we used Multilabel Synthetic Minority Over-sampling Technique (MLSMOTE) (Charte et al., 2015) w.r.t the team labels with minimal data representation.

## 4 Model

The multi-team-labels defect classification task in this research can be summarized with S as the tuple set. $d_i$ and $t_i$ represents the $i^{th}$ defect denoted as D and its corresponding team-labels denoted as T. N, n and m are the total number of defects, the length of the $i^{th}$ defect text and the number of teams-labels of the $i^{th}$ document, respectively.

$$S = \{(d_i, t_i)\}_{i=1}^{N}, D = \{d_i | d_i = \{d_1, d_2, , d_n\}\},$$
$$T = \{t_i | t_i = \{t_1, t_2, , t_m\}\}$$

Our framework, DEFTri aims at assigning team-labels to its corresponding defects based on the conditional probability P($t_i$|$d_i$).

| Defect Text Corpus (Anonymized Excerpts) |
|---|
| ...For a store only query like XXX i am seeing available for scheduled pickup as the stack title on FE when i don't have a slot booked.This stack title should just reflect the XXX query like ios and web..Incorrect XXX mapping (number mapped to XXX.. |
| ...I cant add XXX to my cart from order details from my previous canceled order. There is no actionable CTA.There is an add to cart CTA for the XXX. See attached video. Using ios XXX... |

Table 1: Samples of Defect Text Corpus.

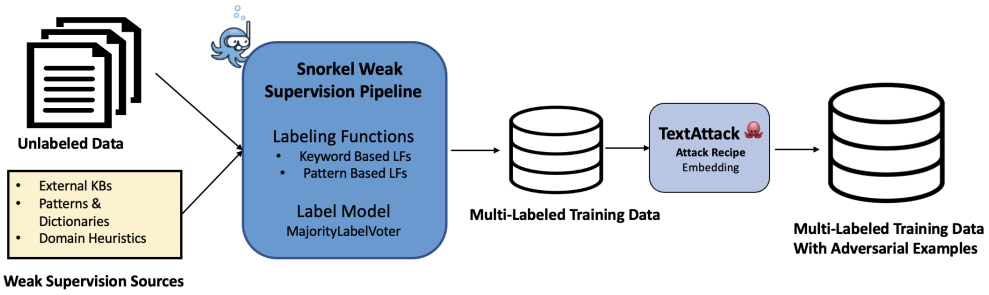


Figure 1: DEFTri Data Generation Methodology

| Rule | corpus->labels (Anonymized) |
|---|---|
| Keyword | 'android' or 'ios' -> [Team-LabelA] |
| Pattern | '*search*' -> [Team-LabelB, Team-LabelC] |

Table 2: Example LFs For Snorkel pipeline

### 4.1 Pre-Trained Model

For our fine-tuning, we use BERT pre-trained transformer embedding from Hugging Face's Transformers library (Wolf et al., 2020).BERT base uncased embeddings are case insensitive and are pre-trained on the English language self-supervised using two objectives - masked language modeling (MLM) and Next Sentence Prediction (NSP). These embeddings were introduced in the original BERT (Devlin et al., 2018) paper and serve as baseline embeddings for our models.

### 4.2 Approach

For our DEFTri framework, we propose 2 novel implementations to derive superior contextual representations from product defect text, that help in improved multi-label defect classification task. We denote the defect corpus(title and description) tokens as $D_i$ and their corresponding token embeddings as $E_{Di}$, where K is the total number of words in the input defect and $D_K$ represents the last token. Similarly, let $L_j$ be the team label text of the $j^{th}$ team of the overall 15 teams, corresponding to the defect corpus. Finally, we derive the positional embedding using BERT and apply classification layer with activation to the last layer of the hidden state at the [CLS] token.

#### 4.2.1 Label Fused Model with [SEP]

We utilize the sentence pair configuration of BERT for text input. We concatenate the team labels text as Sentence A and concatenate the Defect title and description text as Sentence B, both separated by a [SEP] token. Refer Figure 2

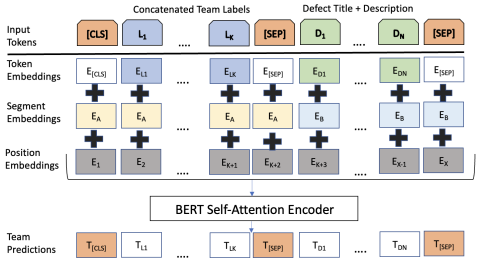


Figure 2: DEFTri LabelFuse Model with [SEP]

#### 4.2.2 Label Fused Model without [SEP]

For our second implementation, we concatenate the team labels text along with Defect title and description text as a single Sentence A, without any [SEP] token as input. Refer Figure 3

| Defect Text (Anonymized) | Adversarial Defect Text (Anonymized) |
|---|---|
| Price showing inconsistently | Price displaying inconsistently |
| Final cost by weight not showing on search tiles | Final prices by weight not showing on search tiles |
| Spacing on Nutrition Label is too large | Spacing on Nourishment Label is too large |

Table 3: Sample cases of Defect text vs Adversarial Defect Text.

| Model | Macro-F1 | Accuracy |
|---|---|---|
| BERT+Linear | 0.8123 | 0.8134 |
| BERT+BiLSTM | 0.8206 | 0.8216 |
| BERT+LabelFuse w/o [SEP]+Linear | 0.8144 | 0.8153 |
| BERT+LabelFuse w/o [SEP]+BiLSTM | **0.8236** | **0.8245** |
| BERT+LabelFuse w [SEP]+Linear | 0.8137 | 0.8150 |
| BERT+LabelFuse w [SEP]+BiLSTM | 0.8229 | 0.8241 |

Table 4: DEFTri Experiments Results For Contextual Multi-TeamLabel Classification on Real Product Defects

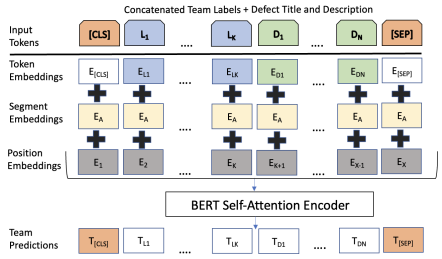


Figure 3: DEFTri LabelFuse Model w/o [SEP]

### 4.3 Classification Head

We experimented with two different dense layers for the classification head - Linear and BiLSTM. Refer Table 5

| Classification Heads | | Dense Layer | | Linear Layer | |
|---|---|---|---|---|---|
| Type | Activation | In | Out | In | Out |
| Linear | Tanh | 768 | 768 | 768 | 15 |
| BiLSTM | ReLU | 768 | 256 | 512 | 15 |

Table 5: DEFTri Classification Head Configurations

### 4.4 Loss Function and Optimizer

For model training we use PyTorch implementation of BCEWithLogitsLoss as our loss function and AdamOptimizer as our optimizer. BCEWithLogitsLoss combines a Sigmoid layer and the Binary Cross Entropy Loss in one single class. In case of multi-label classification the loss can be described as,

$$l_t(x, y) = L_t = \{l_{1,t}, ..., l_{N,t}\}^T,$$
$$l_{n,t} = -w_{n,t}[p_t y_{n,t} \cdot log\sigma(x_{n,t}) + (1 + y_{n,t}) \cdot log\sigma(x_{n,t})]$$

where t=15 and represents the number of team-labels , n is number of sample in the batch and $p_t$ is the weight of the positive answer for team-label t.

### 4.5 Hyper-Parameters

We use a set of hyper-parameters for our experiments. We used manual search for hyper-parameter search and the best model was chosen based on the best top-1 accuracy yielded in the validation data. Refer Table 6

| HParams | Values |
|---|---|
| Dropout | 0.1 |
| Max Sequence Length | 512 |
| Batch-Size | 16 |
| Learning Rate | 1e-5 |
| Weight Decay | 0.01 |
| Adam epsilon | 1e-6 |
| Epochs | 10 |

Table 6: DEFTri Hyper-Parameters

## 5 Experiments

As baseline and our proposed architecture, we use the pre-trained bert-base-uncased model (Wolf et al., 2020; Vaswani et al., 2017). We perform a total of 6 experiments for our models under 3 different settings (1) baseline fine-tuned BERT model with no fused labels (2) fine-tuned BERT with fused labels without [SEP] token and (3) fine-tuned BERT with fused labels with [SEP] token, using 2 classification heads combinations e.g Linear and BiLSTM. Refer Table 4 and Appendix A.1

For data preprocessing step, the corpus is converted to lowercase and tokenzied with one-hot-encoded labels.Our deep learning model is then

trained to predict multiple team-labels for each test sample. At inference time, the model takes in an input of text corpus of defect and predicts a vector of probabilities for each of the 15 team-labels. We used a confidence threshold of 0.55 for our probability vector to obtain a binary vector for comparison with ground-truth.

Measuring accuracy on exact binary vector matching for multi-label classification is too penalizing because of the low tolerance for partial errors. Therefore, we divide our predictions by classes. For each of the team-labels in our dataset, we calculate the number of false positives (FP), false negatives (FN), true positives (TP), true negatives (TN). Finally, to obtain our Accuracy, we sum up the values across each team-labels as below,

$$Accuracy = \frac{\sum TP_t + \sum TN_t}{\sum FP_t + \sum FN_t + \sum TP_t + \sum TN_t}$$

where T=15 and represents the number of team-labels in our dataset and $TP_t$, $TN_t$, $FP_t$ , $FN_t$ represents values of TP, TN, FP, FN for $t^{th}$ team-label. Similarly, we used macro-F1 (F1) scores based on averaged value of precision and recall calculated over all team-labels as below,

$$Precision_t = \frac{TP_t}{FP_t + TP_t}$$

$$Recall_t = \frac{TP_t}{FN_t + TP_t}$$

$$F1 = 2 \times \frac{\frac{1}{T}\sum Precision_t \times \frac{1}{T}\sum Recall_t}{\frac{1}{T}\sum Precision_t + \frac{1}{T}\sum Recall_t}$$

## 6 Analysis

Based on our experiments, we observed that label-fused contextual learning-based fine-tuned BERT models significantly outperformed the base model using only the context of the defect text. The performance boost over the base BERT pre-trained fine-tuned model is because of the context in the label embeddings used in addition to the defect text in the label-fused models, which optimizes on the alignment of features, which makes it possible to classify better. Our team labels were short meaningful English words vs abbreviations which made fused embeddings better for classification when paired as a sentence with the defect texts as inputs. We observed that label-fused model without [SEP] token performed better that with [SEP] token which could have been because of the unnatural formation of Sentence A, where a bunch of team labels are concatenated together.

Also, with the addition of synthetically generated data using adversarial examples for model training, we achieved an average accuracy improvement of 2.69% across our models vs. using the original data only. However, during our experiments we observed that the performance was sensitive towards the choice of text corpus sequence length and perturbation percentage for data augmentation made, during model training. A higher percentage of perturbations combined with a lower sequence length of text corpus negatively impacted performance.

## 7 Future Work

Fine-tuning language models with weak supervision definitely solves the challenge of low labeled data availability. However, the models performance definitely suffers from error-propagation of pseudo-labels generated during the process. Recent research in contrastive self-regularized self-training approach (Yu et al., 2020) and GAN-BERT in adversarial setting (Croce et al., 2020) have shown promising results for fine-tuning BERT-based language models with weak supervision. Also, Contrastive learning and Adversarial Learning approaches applied to various NLP tasks have demonstrated improvement over fine-tuning on BERT-based models (Mohanty et al., 2021; Pan et al., 2021). To further our research, we would improve upon these approaches.

## 8 Conclusion

In this work, we proposed a novel framework, DEFTri for automated defect triage using contextual representations of human-generated defect reviews at Walmart. We discussed our methodology of generating a new proprietary labeled dataset by using weak supervision and adversarial learning, in a few shot setting. We presented two label-fused model approaches for fine-tuning pre-trained BERT. As hypothesized, the experimental results show that our approach improves the multi-label text classification task for defect triage. We also proposed our future work of implementing contrastive learning for fine-tuning using weak supervision.

## Acknowledgments

The author would like to thank colleagues in the Omni Customer Experience org. at Walmart Global

Tech for all their support and encouragement.

## References


Francisco Charte, Antonio J. Rivera, María J. del Jesus, and Francisco Herrera. 2015. Mlsmote: Approaching imbalanced multilabel learning through synthetic instance generation. *Knowledge-Based Systems*, 89:385–397.

Christopher A. Choquette-Choo, David Sheldon, Jonny Proppe, John Alphonso-Gibbs, and Harsha Gupta. 2019. A multi-label, dual-output deep neural network for automated bug triaging. *CoRR*, abs/1910.05835.

Danilo Croce, Giuseppe Castellucci, and Roberto Basili. 2020. GAN-BERT: Generative adversarial learning for robust text classification with a bunch of labeled examples. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2114–2119, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Xinshuai Dong, Luu Anh Tuan, Min Lin, Shuicheng Yan, and Hanwang Zhang. 2021. How should pre-trained language models be fine-tuned towards adversarial robustness? *CoRR*, abs/2112.11668.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks.

Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Andrei Calian, and Timothy A. Mann. 2021. Improving robustness using generated data. *CoRR*, abs/2110.09468.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is BERT really robust? natural language attack on text classification and entailment. *CoRR*, abs/1907.11932.

Ahmed Lamkanfi, Javier Pérez, and Serge Demeyer. 2013. The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 203–206.

Huiting Liu, Geng Chen, Peipei Li, Peng Zhao, and Xindong Wu. 2021. Multi-label text classification via joint learning from label embedding and label correlation. *Neurocomputing*, 460:385–398.

Senthil Mani, Anush Sankaran, and Rahul Aralikatte. 2018. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. *CoRR*, abs/1801.01275.

Ipsita Mohanty, Ankit Goyal, and Alex Dotterweich. 2021. Emotions are subtle: Learning sentiment based text representations using contrastive learning. *CoRR*, abs/2112.01054.

John X. Morris, Eli Lifland, Jin Yong Yoo, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks in natural language processing. *CoRR*, abs/2005.05909.

Lin Pan, Chung-Wei Hang, Avirup Sil, Saloni Potdar, and Mo Yu. 2021. Improved text classification via contrastive adversarial training. *CoRR*, abs/2107.10137.

Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *CoRR*, abs/1711.10160.

Shijing Si, Rui Wang, Jedrek Wosik, Hao Zhang, David Dov, Guoyin Wang, Ricardo Henao, and Lawrence Carin. 2020. Students need more attention: Bert-based attentionmodel for small data with application to automaticpatient message triage. *CoRR*, abs/2006.11991.

Behzad Soleimani Neysiani, Seyed Morteza Babamir, and Masayoshi Aritsugi. 2020. Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. *Information and Software Technology*, 126:106344–106363.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing.

Hongrun Wu, Yutao Ma, Zhenglong Xiang, Chen Yang, and Keqing He. 2021. A spatial-temporal graph neural network framework for automated software bug triaging. *CoRR*, abs/2101.11846.

Yijin Xiong, Yukun Feng, Hao Wu, Hidetaka Kamigaito, and Manabu Okumura. 2021. Fusing label embedding into bert: An efficient improvement for text classification. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, pages 1743–1750. Association for Computational Linguistics.

Yue Yu, Simiao Zuo, Haoming Jiang, Wendi Ren, Tuo Zhao, and Chao Zhang. 2020. Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. *CoRR*, abs/2010.07835.

# A Appendix

## A.1 Experiment Setting

We ran all our experiments on a Google Cloud Platform using a n1-standard-16 machine with NVIDIA Tesla V100 GPUs.