IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 July 2024, accepted 14 August 2024, date of publication 28 August 2024, date of current version 17 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3451125

RESEARCH ARTICLE

A Comparative Study of Contemporary Learning Paradigms in Bug Report Priority Detection

EYÜP HALIT YILMAZ[1], İSMAIL HAKKI TOROSLU[1], AND ÖMER KÖKSAL[2]
[1]Department of Computer Engineering, Middle East Technical University, 06800 Ankara, Türkiye
[2]University of Doha for Science and Technology, Doha, Qatar

Corresponding author: Ömer Köksal (omer.koksal@udst.edu.qa)

This work was supported by Qatar National Library.

ABSTRACT The increasing complexity of software development demands efficient automated bug report priority classification, and recent advancements in deep learning hold promise. This paper presents a comparative study of contemporary learning paradigms, including BERT, vector databases, large language models (LLMs), and a simple novel learning paradigm, contrastive learning for BERT. Utilizing datasets from bug reports, movie reviews, and app reviews, we evaluate and compare the performance of each approach. We find that transformer encoder-only models outperform in classification tasks measured by the precision, recall, and F1 score transformer decoder-only models despite an order of magnitude gap between the number of parameters. The novel use of contrastive learning for BERT demonstrates promising results in capturing subtle nuances in text data. This work highlights the potential of advanced NLP techniques for automated bug report priority classification and underscores the importance of considering multiple factors when developing models for this task. The paper's main contributions are a comprehensive evaluation of various learning paradigms, such as vector databases and LLMs, an introduction of contrastive learning for BERT, an exploration of applicability to other text classification tasks, and a contrastive learning procedure that exploits ordinal information between classes.

INDEX TERMS Bug triaging, contrastive learning, machine learning, natural language processing, software bug report classification, software engineering.

I. INTRODUCTION

The rapid proliferation of software applications has led to an exponential increase in bug reports generated by developers, testers, and users. Manual classification or prioritization of these bugs is a time-consuming process that can be significantly improved through the use of automated methods. In recent years, significant advancements in natural language processing (NLP) techniques have been applied to text data analysis. This can help automate several steps of software development and maintenance processes [1], [2]. The potential benefits of these NLP models in classifying software bugs are clear: they can accelerate the prioritization process, improve team productivity by directing resources toward high-impact issues, and ultimately lead to better-quality software products. However, there is a need for an efficient automated method that considers both contextual information from the bug reports and learned domain knowledge of bug report priorities to classify them into priority categories accurately. Fortunately, recent advancements in natural language processing (NLP) techniques have provided a way forward, essentially compressing information about a domain in the weights of a neural network. With NLP models able to analyze text data and learn representative deep features, it is now possible to exploit this information to classify bug reports into priority categories based on contextual information and semantic relationships between different bug reports. This can accelerate the prioritization process, improve team productivity by directing resources toward high-impact issues, and ultimately lead to better-quality software products. The landscape of deep neural networks for text classification is vast and ever-expanding, with new models entering the scene each day. The question of which paradigm best suits an efficient automated priority detection method still stands. We aim to shed light on the answer to this question with a pioneering comparative study in which we experiment with various contemporary deep learning techniques such as transformer encoder classifiers

The associate editor coordinating the review of this manuscript and approving it for publication was Filbert Juwono.

(e.g., BERT[1]), vector databases,[2] Large Language Models and Retrieval Augmented Generation (RAG)[3] for automated prioritization of software bug reports.

In this paper, we present a comprehensive study that evaluates the performance of these various deep learning models using a bug report dataset obtained from the open-source Mozilla project. The structure of the problem leverages a collection of different sources, including bug report repositories, movie reviews, and app store ratings. We aim to understand how each approach captures contextual information relevant to bug classification tasks and contribute to a better understanding of how deep learning techniques can be leveraged in the automated prioritization of software bugs using NLP approaches.

The novelty of our work is the proposition of contrastive loss for fine-tuning encoder models when the downstream task enables an ordering between different classes. We ask the following research questions:

- Does contrastive learning provide a performance boost in classification tasks with an order present between class labels? (RQ1)
- How do generative language models without fine-tuning perform classification tasks that require domain-specific knowledge? (RQ2)
- Can Retrieval Augmented Generation (RAG) improve the classification performance of LLMs? (RQ3)
- How much does the domain knowledge requirement affect the downstream performance of pre-trained models? (RQ4)

The proposed work is organized as follows: Section II provides an overview of the related literature on NLP and bug report analysis, Section III introduces contrastive learning for BERT, and Section IV provides the alternative methods presented in our paper, which includes dataset preparation, experimental setup, evaluation metrics, and a comparative study of various deep learning models used in this research. Section V presents the experimental design used in the paper. In Section VI, we present and discuss our results while giving insights into their implications for future research directions in automated prioritization of software bugs using NLP techniques. Finally, Section VII concludes the paper with some final remarks on its contributions to the field of natural language processing applied to downstream applications such as bug report analysis.

II. RELATED WORK

Software bug analysis has been an important subject and has been studied extensively [6], [7], [8] Artificial Intelligence and Machine Learning techniques in software development, specifically for bug triaging, are a growing and active field of research [9]. Classification methods are commonly applied in automatic software bug report assessment. Researchers proposed [2] an ML-based approach to predict fault types using textual bug reports from 70+ projects with up to 69% macro average F1 score. They established a baseline for non-expert human performance in debugging tool selection. The models showed inter-project transferability and could aid inexperienced developers by saving time and resources during debugging. In another study, a comparative analysis ([10] is conducted on three Multi-Objective Evolutionary Algorithms (MOEAs) - NSGA-II, NSGA-III, and MOEA/D - to automatically generate finite state machines (FSMs) based on bug reports written in natural language. The results show that the proposed approach significantly outperforms the baseline tool KLFA for real-world software programs with a greater number of bugs detected by NSGA-II while also controlling the model size and avoiding local optima through using all three objectives during evolution (size, over-approximation, and under-approximation). In [11], Bidirectional Encoder Representation from Transformers (BERT) [12] is adapted to the software security domain, with their model outperforming baseline models on both large and small sample datasets, detecting 56 hidden vulnerabilities through deployment in Mozilla and RedHat projects. This demonstrates the practical value of deployed NLP models for automated software improvement processes. Reference [13] proposed a method called MaCa that leverages machine learning techniques to accurately classify action words within bug reports associated with mobile apps. This significantly improves the success rates of existing approaches, such as Yakusu and ReCDroid, in reproducing bugs.

In many cases, data imbalance and data scarcity hinder downstream model performance. Various studies addressed this issue via different techniques, such as oversampling, a well-known approach to generating synthetic data or resampling to adjust the distribution of samples in the training set. The study [14] proposes a priority prediction method using comment intensiveness features and a SMOTE-based data balancing scheme on three open-source projects: Eclipse, Mozilla, and OpenOffice. The proposed approach achieves 0.6078 Precision, 0.4927 Recall, 0.4465 F1-score, and 0.7836 Accuracy in priority prediction with five classification models: Multinomial Naïve Bayes, Support Vector Machines, Random Forest, Extra Trees, and eXtreme Gradient Boosting. The results show that CIS-SMOTE-based models outperform two advanced approaches, eApp and cPur, regarding all performance measures. CLeBPI (Contrastive Learning for Bug Priority Inference) [15] leverages pre-trained language models and contrastive learning to tackle label imbalance problems by automatically learning contextual representations of bug reports without manual feature engineering and shows its effectiveness with an improvement in weighted average F1 score compared to four baseline approaches on a public dataset. Reference [16] experiment with six

[1]Bidirectional Encoder Representations from Transformers (BERT) is a language model based on the transformer architecture. It was introduced in 2018 by Google [3].

[2]A vector database is a database to store vectors (fixed-length lists of numbers) along with other data items aiming to do neighborhood search efficiently [4].

[3]RAG is a popular method that uses a retriever to fetch relevant documents from a large corpus and then feeds these documents into a generative model to produce a final answer or text generation [5].

state-of-the-art rebalancing methods combined with five popular classification algorithms for security bug reports (SBR) predictions and demonstrate the potential of these techniques to improve performance by 267% in the best case and 75% on average, based on a comparative study using real-world projects.

Reference [17] proposed an adaptive ranking approach to profile developers through their commit messages using various features, including domain knowledge encoding and a learning-to-rank technique, to recommend suitable developers to handle bug reports. The model was evaluated on around 22,000 bug reports from four large-scale open-source Java projects, with significant outperformance of state-of-the-art methods regarding the percentage of correctly recommended top developers within the rankings. Contrastive learning [18] improves the model's generalization capability in zero-shot tasks while increasing cross-domain generalization. It is achieved by incorporating label semantics into the training procedure, transforming the classification task into a semantic understanding task. The proposed approach is promising in question-answering and reading comprehension applications.

Priority detection is a fundamental part of the automated bug report assessment process. Earlier efforts in this line of research investigate different aspects of the problem. In [19], the authors propose an automatic approach to predict bug report priorities using convolutional neural networks (CNNs). They apply natural language processing techniques and software engineering domain-specific emotion analysis on textual information of bug reports, converting them into vectors based on their syntactic and semantic relationship. The proposed method outperforms state-of-the-art approaches by more than 24% in average F1-score across multiple open-source projects. For a similar problem, bug report severity detection, [20] proposed a deep neural network-based automatic approach to predicting bug report severities, outperforming state-of-the-art approaches by an average f-measure improvement of 7.90%. The study employed natural language processing techniques for text preprocessing and assigned emotion scores before passing the constructed vector and score through a deep neural network classifier for prediction purposes.

Our literature survey indicates that although bug report analysis is a well-studied area, detecting the priority level from the report's textual content is still not well explored. To the best of our knowledge, applying contrastive learning to this task is a novel idea.

In our work, we do not conduct a detailed literature survey to the full extent, as it exceeds the scope of the paper. Yet, we examine the state-of-the-art methods in multiple bug report priority detection paradigms with detailed analyses.

III. CONTRASTIVE LEARNING FOR BERT

Software bug report priority detection task has a unique feature in its formulation as a classification task that can be exploited with a dedicated learning procedure. Each software bug report is labeled with a priority level, i.e., P1, P2, P3, P4, or P5, which indicates an ordering between the bug reports in terms of priority. In a deep representation learned by a neural network, P1 and P2 labeled bug reports would be expected to map to similar or closer embeddings to each other than P1 and P5 labeled bug reports. We employ a contrastive learning loss to incorporate this order information into the training process to enhance modeling capabilities. The essential observation is as follows: During training, when the model prediction class for a given bug report differs significantly from the ground truth label, for instance, model prediction is P1 and the ground truth label is P5, the loss function should yield a greater value than a closer prediction, model prediction is P1 and the ground truth label is P2. This distinction is not present in the commonly used cross-entropy loss, where each class is considered independently. Based on this observation, we propose using a contrastive learning approach for fine-tuning BERT with a loss function tailored explicitly for such a classification task. So, we modify the training loss function as

$$\begin{aligned}\text{Contrastive Loss} = \text{Cross-Entropy Loss} + \lambda \\ \times \text{Mean Squared Error}\end{aligned} \tag{1}$$

$$\text{Cross-Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) \tag{2}$$

$$\text{Mean Squared Error} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3}$$

where N is the number of classes, y_i is the ground truth probability for class i, $\hat{y}_i$ is the predicted probability for class i, and λ is a hyperparameter that controls the weight of the mean squared error term relative to the cross-entropy loss. We propose to add mean squared error (MSE) as a regulation term to the cross-entropy loss function.

The contrastive loss function aims to have the best of two worlds: the cross-entropy term optimizes the classification performance, while the mean squared error term regularizes the learning procedure to achieve closer embeddings for similarly prioritized bug reports. To fine-tune the effect of the regularization term, we use a hyperparameter λ.

Another serious problem in software bug report priority detection is a severe imbalance in training data. We modify the loss function with a balancing term that exaggerates the loss values for poorly represented classes, e.g., P4 and P5. We calculate the class distributions beforehand and use the percentage values as weights of the balancing term as given in 4.

$$\text{Imbalance Loss} = \frac{1}{N}\sum_{i=1}^{N} w_j \cdot y_i \cdot log(p(y^{(i)}|x)) \tag{4}$$

Here, w_j corresponds to the class weight calculated from the entire training dataset. We apply a mask to filter out correct predictions in every batch and apply the imbalance term only to wrong predictions. The proposed learning

method amplifies the learning signal for instances from rare classes when the model makes mistakes.

Finally, the two loss functions are combined by simple addition to constitute the loss function as given in 5.

$$\text{Loss Function} = \text{Contrastive Loss} + \text{Imbalance Loss} \quad (5)$$

$$\begin{aligned}\text{Loss Function} = &-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) \\ &+ \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\ &+ \frac{1}{N}\sum_{i=1}^{N} w_j \cdot y_i \cdot log(p(y^{(i)}|x)) \end{aligned} \quad (6)$$

IV. ALTERNATIVE METHODS

The technical capabilities of neural language models have increased remarkably in many downstream applications. With the release of ChatGPT and its open-source counterparts, such as LLaMA [21] and Llama 2 [22], the interface for human interaction with large language models (LLMs) transformed into natural conversation. The method that enabled this transformation is commonly referred to as instruction tuning. Reinforcement Learning from Human Feedback (RLHF) [23] provides a preference-based learning procedure in contrast to regular self-supervised fine-tuning. RLHF is a human-in-the-loop alternative in settings with no training data available.

LLMs' ubiquitous data generation capabilities [24] triggered an exponential growth in instruction fine-tuned models. OpenChat [25], Mistral7B [26], and many others enabled general-purpose chatbot applications to be run locally in commercial settings. The training domain of these models is usually not restricted; they are general-purpose chat models.

However, despite their impressive capabilities, LLMs still face some challenges that need to be addressed before they can effectively support bug report priority detection tasks. Alignment is a significant component in ensuring model outputs follow a desired format. Consequently, many techniques emerge that aim to tailor model responses according to the needs of the downstream task. Prompt engineering [27], [28] is a method to maximize the occurrence of the desired tokens at the model output by changing the input structure to fit specific criteria the model prefers. These criteria might be set by the pretraining data, hyperparameters of text generation, and other factors that are not readily available for examination. Therefore, each downstream task and application needs a dedicated engineering effort to increase performance.

An emergent problem in LLM applications is hallucination, i.e., the tendency of the model to generate confident yet incorrect factual outputs. To mitigate this issue, Retrieval Augmented Generation (RAG) [5] adds a retrieval component between the user input and the LLM [29]. RAG first applies a user query search and provides the retrieved information to LLM to assist the generation process. Effectively, RAG constitutes a means to reduce hallucination [30] by modifying the LLM prompt to include factual information and helping the model depend on the retrieved documents.

With the increasing embedding capabilities of neural models, the idea of transforming the storage of documents into a storage of embeddings became feasible. Vector databases are proposed to provide a storage medium that operates on the semantic relationship between vector embeddings according to the contents of the data used to generate the embedding. For natural language inputs, a language model first generates a sequence embedding for the given input, such as a document. Then, the vector database stores the document based on a similarity measure of its embedding and existing embeddings in the database. The input data can be in any form acceptable by the embedding model, such as audio, image, or text. This provides an advantage in the retrieval step, where one can apply a query to the system in any form.

We harness these technologies in software bug report priority detection with certain modifications. We apply few-shot inference with $k = 15$ demonstrations in the system prompt. Figure 1 illustrates the framework we use for priority detection. We use OpenChat [25] in our experiments.

The LLM output is decoded into text via greedy sampling, as given in 7. The token generated at each step is the most probable in the Softmax output of the transformer decoder. Multiple tokens may be utilized to generate the required class label. If the output does not match class labels, we re-run the generation process with a different seed.

$$\begin{aligned} p(t_1, \ldots, t_{T-1}|X) &= \prod_{i=0}^{n} \max_{j=1}^{|V|} P(v_j|v_{\leq i})y \\ &= argmax_{x \in V^T} p(x|X) \end{aligned} \quad (7)$$

A random sampling of the training data in the generation process might bias the model towards dominant classes. Further, the content of the test instance might differ from the training samples used in the few-shot prompt. To improve the quality of the system prompt, we employ a vector database to retrieve the most similar $k = 15$ training instances and construct the system prompt using these instances. This simple RAG system uses a similarity-based retrieval step before the generation step. The retrieved instances are expected to have similar labels and condition the model to generate the correct prediction label.

V. EXPERIMENTAL DESIGN

The literature attempts to determine the classification difficulty of textual datasets by combining different parameters and statistics. One such study, Evolutionary Data Measures (EDM) [31], utilizes evolutionary algorithms for feature selection over 27 datasets. In this study, we use the provided tool to measure the classification difficulty of our datasets.

Using the title and description fields, we utilize the Mozilla Core dataset from [32] for bug report priority detection. We assess the statistical properties of this dataset with the

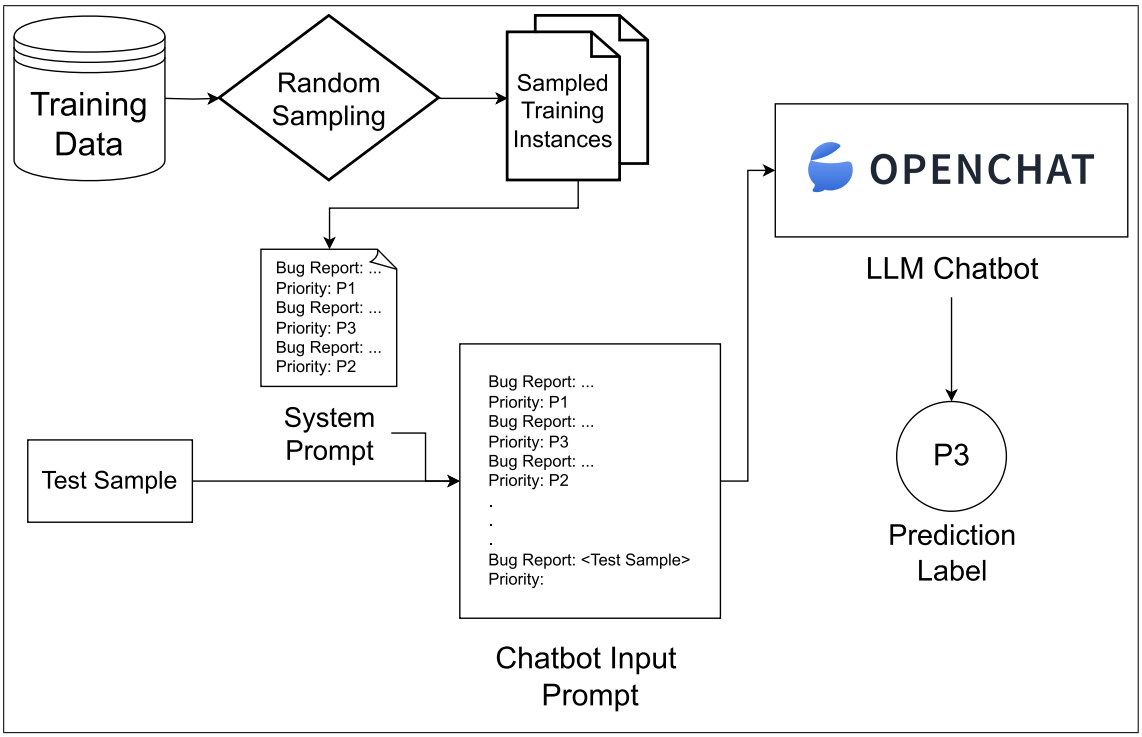


FIGURE 1. Framework for the chatbot application in bug report priority detection.

TABLE 1. Mozilla dataset difficulty report.

Metric	Value	Comment
Dataset Size	40565	-
Vocab Size	288751	-
Number of Classes	5	-
Mean Items Per Class	8113.0	-
Min. Items in a Class	689	EXTREMELY LOW
Average Sentence Length	862.117	-
Distinct Words : Total Words	0.068	GOOD
Class Imbalance	0.802	SOMEWHAT HIGH
Class Diversity	1.102	GOOD
Max. Hellinger Similarity	0.731	HIGH
Mutual Information	1.424	GOOD
Difficulty	4.127	HIGH

EDM tool and report the results in Table 1. The severe imbalance problem is apparent in the minimum number of instances in a class (P5 priority class in this dataset), which hinders the model's ability to learn the descriptive features of this class. The vocabulary of the dataset is diverse, as indicated by the 0.068 score, and class diversity is beneficial for the classifiers. However, high-class imbalance and Hellinger similarity yield an overall HIGH difficulty score of 4.127.

We select two new datasets from similar tasks to assess the generalization capabilities of the methods we employ in other domains. We ensure the labels in the classification tasks contain an order of information concerning each other, just as priority labels do. We use an app review dataset from [33] and Rotten Tomatoes movie reviews dataset [34]. The target labels are the ratings from users and audiences on a 1-5 rating scale. Naturally, there is an ordering information between classes similar to priority labels in software bug report priority detection, which can be exploited via contrastive learning.

We apply the same EDM difficulty assessment procedure to the app and movie review datasets and present their results in Tables 2 and 4 respectively. The class imbalance problem is not apparent in the app reviews dataset; there is a slight problem with class diversity and maximum Hellinger similarity, but mutual information between classes is not problematically high. The overall assessment for classification difficulty is SOMEWHAT HIGH for this dataset, with a 3.805 difficulty score.

The Rotten Tomatoes movie reviews dataset is relatively small, given the number of instances, and the number of instances in the smallest class is slightly low. The class imbalance is not high, but class diversity and maximum Hellinger similarity increase the difficulty level. The difficulty score for this dataset is 4.148, and the assessment is HIGH. We expect classifiers to struggle with achieving high classification scores.

We report classification results in detail concerning precision, recall, and F1 score for each method and each class. We avoid averaging over classes because over-fitting models due to severe class imbalance tend to predict the majority class for every test case and achieve moderate mean scores.

TABLE 2. App reviews dataset difficulty report.

Metric	Value	Comment
Dataset Size	30000	-
Vocab Size	41619	-
Number of Classes	5	-
Mean Items Per Class	6000.00	-
Min. Items in a Class	6000	GOOD
Average Sentence Length	180.945	-
Distinct Words : Total Words	0.048	GOOD
Class Imbalance	0.0	GOOD
Class Diversity	1.609	SOMEWHAT HIGH
Max. Hellinger Similarity	0.804	HIGH
Mutual Information	1.344	GOOD
Difficulty	3.805	SOMEWHAT HIGH

TABLE 3. Rotten tomatoes tomatometer rating difficulty report.

Metric	Value	Comment
Dataset Size	7291	-
Vocab Size	24407	-
Number of Classes	5	-
Mean Items Per Class	1458.2	-
Min. Items in a Class	390	SLIGHTLY LOW
Average Sentence Length	160.495	-
Distinct Words : Total Words	0.137	HIGH
Class Imbalance	0.579	GOOD
Class Diversity	1.379	SOMEWHAT HIGH
Max. Hellinger Similarity	0.693	SOMEWHAT HIGH
Mutual Information	1.360	GOOD
Difficulty	4.148	HIGH

TABLE 4. Setup used in the experiments.

Parameter	Value
CPU	Intel Xeon E5 (24 Core)
GPU	Nvidia RTX 2080Ti x4
GPU-RAM	12 GB x4
Operating System	Ubuntu 18.04
System RAM	64 GB

We apply 5-fold cross-validation in every experiment. The test splits in different folds do not have overlapping instances to ensure that the observations are independent. The reported scores are averages over 5 folds.

A. EXPERIMENTAL SETUP

In our study, we conducted our computations using a single-board computer. The detailed specifications of the experimental setup, which were consistently utilized in all our experimental procedures, can be found in the table below.

Our study has a number of hyperparameters, which we report here for reproducibility. All models are fine-tuned for 5 epochs. Adam optimizer [35] is used with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Learning rate is specified as $lr = 5 \times 10^{-5}$. We use linear scheduler for learning rate scheduling.

LLM inference in the OpenChat methods implements greedy decoding with softmax temperature $\tau = 1.0$. The vector database implementation in RAG + OpenChat method utilizes the all-MiniLM-L6-v2[4] model for encoding, which is optimized over sequence pairs to minimize cosine distance for similar sequences [36].

VI. RESULTS AND DISCUSSION

Table 5 presents the software bug report priority detection results for all methods. Several observations regarding different aspects of the problem are made.

- **Contrastive BERT outperforms other methods in most cases.** Concerning all three performance metrics and in five different classes, the Contrastive BERT approach outperforms standard BERT, OpenChat LLM, and RAG approach in 12 out of 15 cases (5 classes and 3 metrics). RAG improves precision for P3, the most dominant class, without a better F1 score, which may indicate an over-fitting problem. This observation indicates that a carefully tailored classifier model can outperform models that are orders of magnitude larger in many parameters. Incorporating the domain information about order relations between classes improves modeling capability compared to the standard BERT model. For our RQ1, this experimental result indicates that contrastive learning is an effective fine-tuning paradigm that boosts classification performance.
- **P3 class performance is higher than other classes for all methods.** Severe imbalance in the dataset causes model predictions to prefer the dominant class for fine-tuning approaches and few-shot LLM outputs. The imbalance loss term in the Contrastive BERT approach aims to mitigate the adverse effects of dataset imbalance by calculating class distributions in the training set. Recall values are higher than precision values in BERT-based methods, which indicates false alarms in the P3 class.
- **LLM transformer decoder underperforms in classification.** The OpenChat model, based on transformer decoder architecture for text generation, fails to generalize from its training domain to the software domain without further fine-tuning training. Few-shot demonstrations acquired through random sampling do not suffice for conditioning the model to generate a correct class label. As an answer to our RQ2, generative LLM decoders without fine-tuning are inferior in performance to fine-tuned encoder classifiers.
- **RAG improves LLM performance in most cases.** Integrating a vector database for RAG assistance in LLM generation increases classification performance, specifically according to the F1 score metric.

[4]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

TABLE 5. Per-class classification performances on mozilla core dataset.

	BERT					Contrastive BERT				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
Precision	40.32	35.73	74.25	10.70	15.40	**45.61**	**37.30**	76.12	**14.31**	**20.70**
Recall	34.18	31.92	**82.18**	6.27	6.42	**37.51**	**35.35**	82.07	10.40	**10.60**
F1-Score	36.76	33.52	77.93	7.78	8.54	**40.35**	**35.99**	**78.85**	**11.46**	**12.81**
	OpenChat					RAG+OpenChat				
Precision	18.46	17.42	61.54	3.35	2.05	15.81	9.92	**83.62**	5.89	4.35
Recall	14.41	30.90	46.11	9.40	0.39	31.51	16.98	67.28	**12.79**	0.86
F1-Score	16.19	22.28	52.72	4.94	0.66	21.05	12.52	74.57	8.07	1.44

TABLE 6. Statistical significance test results for mozilla core dataset.

	Precision		
RAG + OpenChat	BERT	Contrastive BERT	OpenChat
	<1%	<1%	<1%
	Recall		
BERT	Contrastive BERT	OpenChat	RAG + OpenChat
	>5%	<1%	<5%
	F1 Score		
RAG + OpenChat	BERT	OpenChat	RAG + OpenChat
	<5%	<5%	<1%

TABLE 7. Statistical significance test results for app reviews dataset.

	Precision		
Contrastive BERT	BERT	OpenChat	RAG + OpenChat
	>5%	<1%	<1%
	Recall		
Contrastive BERT	BERT	OpenChat	RAG + OpenChat
	>5%	<1%	<1%
	F1 Score		
Contrastive BERT	BERT	OpenChat	RAG + OpenChat
	>5%	<1%	<1%

TABLE 8. Statistical significance test results for rotten tomatoes tomatometer rating dataset.

	Precision		
OpenChat	BERT	Contrastive BERT	RAG + OpenChat
	<5%	<1%	<1%
	Recall		
BERT	Contrastive BERT	OpenChat	RAG + OpenChat
	>5%	<1%	<1%
	F1 Score		
BERT	Contrastive BERT	OpenChat	RAG + OpenChat
	>5%	<1%	<1%

A similarity-based retrieval system utilizes a transformer encoder architecture to generate semantic embeddings [37] and help prepare a more suitable prompt for the LLM in the few-shot setting [38]. This result provides an affirmative answer to our RQ3.

- **P4 and P5 class performances are distinguishably lower than other classes.** The training dataset does not include many P4 and P5 class samples, which leads to underrepresentation during fine-tuning training for BERT-based methods. Notably, the few-shot generative model also fails to predict P4 and P5 labeled instances accurately.

Transformer decoder classifiers outperform generative LLM models in automated software bug report priority prediction. The task domain requires the model to have a sophisticated understanding of the software management process, which is better incorporated via parameter updates with fine-tuning. Contrastive learning provides a better exploitation of the ordering between classes, boosting classification performance for all classes.

We also test how each method performs in app and movie review domains. Table 9 reports classification performances in each class. We report classes with the same labels for consistency, i.e., from P1 to P5. Table 9 shows that contrastive learning for BERT does not yield a consistent advantage over regular fine-tuning in the app reviews domain. An underlying reason might be that there is a clear distinction concerning different scores attached to review texts apparent in the tone and vocabulary. This is not the case for software bug reports since they usually contain code snippets, error logs, and software-related terminology, which makes it harder for models to distinguish priority labels.

LLM performance in the app reviews domain improves with RAG in most cases, but fine-tuned transformer encoder classifiers outperform few-shot inference of transformer decoders. P1 and P5 class performances are noticeably higher for all models, specifically for LLMs. This can be attributed to a bias in user behavior that favors extremes, either full praise or a full complaint - leading to more reviews with extreme ratings (P1 and P5), as opposed to neutral ones.

Table 10 shows classification results for the movie reviews domain. LLM-based approaches outperform transformer

TABLE 9. Per-class classification performances models on app reviews dataset.

	BERT					Contrastive BERT				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
Precision	59.95	**40.87**	39.52	**50.36**	69.78	**61.08**	39.75	**40.34**	50.06	69.91
Recall	**56.07**	45.32	**45.33**	41.77	**66.29**	52.80	**46.76**	42.89	47.00	65.89
F1-Score	**57.88**	42.72	**41.96**	45.24	67.63	56.39	**42.91**	41.42	**48.36**	**67.75**
	OpenChat					RAG + OpenChat				
Precision	54.70	35.69	35.01	41.12	71.85	55.22	36.71	35.73	41.46	**77.61**
Recall	43.13	38.77	37.48	**51.41**	57.79	48.88	41.79	39.61	50.63	50.31
F1-Score	48.23	37.17	36.21	45.69	64.04	51.86	39.08	37.57	45.59	61.05

TABLE 10. Per-class classification performances models on rotten tomatoes tomatometer rating dataset.

	BERT					Contrastive BERT				
	P1	P2	P3	P4	P5	P1	P2	P3	P4	P5
Precision	50.15	54.47	34.15	31.59	48.67	50.65	56.28	34.36	30.80	49.24
Recall	35.73	43.74	44.34	29.57	55.74	36.46	38.39	41.81	35.23	56.14
F1-Score	41.73	48.52	38.58	30.55	51.96	42.40	45.64	37.72	32.87	52.46
	OpenChat					RAG + OpenChat				
Precision	**80.35**	45.49	37.12	**66.56**	**50.02**	43.36	**62.68**	**39.17**	31.48	48.88
Recall	**45.66**	**71.73**	60.84	21.88	**73.46**	42.86	57.49	**62.63**	**47.54**	23.13
F1-Score	**58.23**	55.67	46.11	32.93	**59.52**	43.11	**59.97**	**48.20**	**37.88**	31.41

encoder classifiers. In this case, contrary to software and app review domain observations. For RQ4, we can say that the domain of movie review classification is less technical than the other datasets, especially the software bug dataset, which provides an advantage in zero-shot inference for LLMs trained on unconstrained corpora.

Further, we observe that RAG is not uniformly helpful. It provides a performance boost according to the F1 score in some classes while deteriorating it in extreme cases (P1 and P5). This might indicate that randomly sampled instances in the few-shot setting tend to favor the extremes. In contrast, retrieval of similar samples shifts the distribution towards a centralized shape (P3 achieves the highest recall, and P2 achieves the highest F1 score).

We verify the statistical significance of our findings using a student's t-test for every pair in each metric. Namely, we calculate the average performance over all classes for a given experiment and performance metric and mark the winner method. Then, we apply a t-test (using the SciPy implementation [39]) between the winner and all other methods. We report the results of the statistical significance tests in Tables 5,6, and 7 for the Mozilla Core dataset, App Reviews dataset, and Rotten Tomatoes Tomatometer Rating Dataset, respectively. The tables report that the measured p-values are less than 1%, less than 5%, but more than 1%, or higher than 5%. We observe that most of the results are statistically significant.

When comparing the results of three different datasets, it is evident that the best results are consistently obtained from the movie review dataset, regardless of the methodology used. Conversely, the worst results are observed with the bug report dataset. Our observations behind these outcomes are as follows:

- Movie reviews are written in high-quality English with very few grammatical mistakes compared to the other 2 datasets.
- Bug reports contain very short, informal natural language texts, often followed by long bug dumps with complex and formal symbolic texts.

- App reviews are also written informally and tend to have more grammatical errors than movie reviews.

It is clear that the quality of grammar and language significantly impacts the success of any methodology employed.

VII. CONCLUSION

Software bug priority detection provides an automated assessment given the textual input of bug reports. We explore transformer encoder classifiers (BERT-based methods) and Large Language Models (LLMs) using the transformer decoder architecture for text generation. Fine-tuning pre-trained encoders tend to perform better than few-shot inference with LLMs, even though there is an order of magnitude difference between the number of parameters and model size in favor of LLMs. Retrieval Augmented Generation (RAG) improves LLM performance, where a vector database retrieves samples similar to a given query. The vector database utilizes a transformer encoder (SBERT) under the hood, which lets the system harness the embedding space.

Contrastive learning provides a better way to incorporate domain knowledge into the model training process by leveraging the order information between classes. Wrong predictions further away from the ground truth are penalized more to regularize the model during training. Imbalance loss aims to negate the disadvantage of poorly represented classes by amplifying the loss values according to class distributions in the data. This way, the network can better focus on the minority classes that are underrepresented and harder to detect compared to the majority ones.

Software bug reports domain requires expert-level knowledge from models, which is hard to obtain in pretraining. Other more widely available domains, such as app reviews and movie reviews, highlight suitable applications of different models. Based on the problem's nature and the data's structure, contrastive learning can help mitigate predictive bias toward edge cases and extremes. At the same time, few-shot LLM inference and RAG systems improve classification performance for other classes.

The answers to our research questions can be given briefly as follows:

- RQ1: Contrastive learning provides a performance boost in classification tasks when there is an order between class labels.
- RQ2: Generative LLM decoders without fine-tuning fail to outperform fine-tuned transformer encoders when domain-specific knowledge is essential.
- RQ3: Retrieval Augmented Generation (RAG) improves the classification performance of LLMs when fine-tuning is not an option.
- RQ4: When domain-specific knowledge is more needed, such as in software bug report priority detection, there is a more apparent performance gap, which can be mitigated by contrastive learning.

In future work, we plan to investigate more advanced techniques such as ensemble learning, a mixture of experts, and fine-tuning on LLMs with the help of memory-efficient learning procedures such as Low-Rank Adaptation (LoRA) [40] and QLoRA [41]. Additionally, we will explore alternative representations for software bug reports that exploit the code snippets and error logs in a structured way.

Developing effective software bug priority detection models is critical for ensuring high-quality software products. Our research has shown that transformer encoder classifiers (BERT-based methods) perform well on this task, with fine-tuning pre-trained encoders proving to be more successful than few-shot inference using Large Language Models even though LLMs have significantly more parameters and larger model sizes compared to BERT. We also found that incorporating domain knowledge through contrastive learning improves performance by regularizing the models during training, while imbalance loss addresses class distribution issues for underrepresented classes.

REFERENCES

[1] C. E. Öztörk, E. H. Yilmaz, Ö. Köksal, and A. Koç, "Software module classification for commercial bug reports," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. Workshops (ICASSPW)*, Jun. 2023, pp. 1–5.

[2] T. Hirsch and B. Hofer, "Using textual bug reports to predict the fault category of software bugs," *Array*, vol. 15, Sep. 2022, Art. no. 100189.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[4] J. Johnson, M. Douze, and H. Jegou, "Faiss: A library for efficient similarity search," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 113–122.

[5] P. Levis, E. Perez, A. Piktus, F. Petroni, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W.-T. Yin, T. Rocktaschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 9459–9474.

[6] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.*, vol. 47, no. 2, pp. 145–180, Feb. 2017.

[7] J. Zhang, X. Wang, D. Hoa, B. Xie, L. Zhang, and H. Mei, "A survey on bug-report analysis," *Sci. China Inf. Sci.*, vol. 58, pp. 1–4, Jan. 2015.

[8] H. Fu, Z. Wang, X. Chen, and X. Fan, "A systematic survey on automated concurrency bug detection, exposing, avoidance, and fixing techniques," *Softw. Quality J.*, vol. 26, no. 3, pp. 855–889, Sep. 2018.

[9] R. Bocu, A. Baicoianu, and A. Kerestely, "An extended survey concerning the significance of artificial intelligence and machine learning techniques for bug triage and management," *IEEE Access*, vol. 11, pp. 123924–123937, 2023.

[10] G. Guizzo, F. Califano, F. Sarro, F. Ferrucci, and M. Harman, "Inferring test models from user bug reports using multi-objective search," *Empirical Softw. Eng.*, vol. 28, no. 4, p. 95, Jul. 2023.

[11] X. Cao, T. Liu, J. Zhang, M. Feng, X. Zhang, W. Cao, H. Sun, and Y. Zhang, "SbrPBert: A BERT-based model for accurate security bug report prediction," in *Proc. 52nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2022, pp. 129–134.

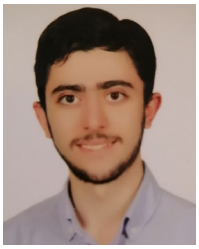
[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics Human Lang. Technol.*, 2019, pp. 4171–4186.

[13] H. Liu, M. Shen, J. Jin, and Y. Jiang, "Automated classification of actions in bug reports of mobile apps," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 128–140.

[14] A.-H. Dao and C.-Z. Yang, "Automated priority prediction for bug reports using comment intensiveness features and SMOTE data balancing," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 33, no. 3, pp. 415–433, Mar. 2023.

[15] W.-Y. Wang, C.-H. Wu, and J. He, "CLeBPI: Contrastive learning for bug priority inference," *Inf. Softw. Technol.*, vol. 164, Dec. 2023, Art. no. 107302.

[16] W. Zheng, Y. Xun, X. Wu, Z. Deng, X. Chen, and Y. Sui, "A comparative study of class rebalancing methods for security bug report classification," *IEEE Trans. Rel.*, vol. 70, no. 4, pp. 1658–1670, Dec. 2021.
[17] B. Alkhazi, A. DiStasi, W. Aljedaani, H. Alrubaye, X. Ye, and M. W. Mkaouer, "Learning to rank developers for bug report assignment," *Appl. Soft Comput.*, vol. 95, Oct. 2020, Art. no. 106667.
[18] Y. Liang, T. Tohti, and A. Hamdulla, "Contrastive classification: A label-independent generalization model for text classification," *Expert Syst. Appl.*, vol. 245, Jul. 2024, Art. no. 123130.
[19] Q. Umer, H. Liu, and I. Illahi, "CNN-based automatic prioritization of bug reports," *IEEE Trans. Rel.*, vol. 69, no. 4, pp. 1341–1354, Dec. 2020.
[20] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, vol. 7, pp. 46846–46857, 2019.
[21] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.
[22] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.
[23] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2022, pp. 27730–27744.
[24] L. Mou, "Search and learning for unsupervised text generation," *AI Mag.*, vol. 43, no. 4, pp. 344–352, Dec. 2022.
[25] G. Wang, S. Cheng, X. Zhan, X. Li, S. Song, and Y. Liu, "OpenChat: Advancing open-source language models with mixed-quality data," 2023, *arXiv:2309.11235*.
[26] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. Le Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed, "Mistral 7B," 2023, *arXiv:2310.06825*.
[27] T. Sorensen, J. Robinson, C. Rytting, A. Shaw, K. Rogers, A. Delorey, M. Khalil, N. Fulda, and D. Wingate, "An information-theoretic approach to prompt engineering without ground truth labels," in *Proc. 60th Annu. Meeting Assoc. for Comput. Linguistics*. Dublin, Ireland: Association for Computational Linguistics, 2022, pp. 819–862.
[28] B. Clavié, A. Ciceu, F. Naylor, G. Soulié, and T. Brightwell, "Large language models in the workplace: A case study on prompt engineering for job type classification," in *Proc. Int. Conf. Appl. Natural Lang. Inf. Syst.* Cham, Switzerland: Springer, 2023, pp. 3–17.
[29] Y. Ahn, S.-G. Lee, J. Shim, and J. Park, "Retrieval-augmented response generation for knowledge-grounded conversation in the wild," *IEEE Access*, vol. 10, pp. 131374–131385, 2022.
[30] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval augmentation reduces hallucination in conversation," in *Proc. Findings Assoc. Comput. Linguistics (EMNLP)*, 2021, pp. 3784–3803.
[31] E. Collins, N. Rozanov, and B. Zhang, "Evolutionary data measures: Understanding the difficulty of text classification tasks," in *Proc. 22nd Conf. Comput. Natural Lang. Learn.*, 2018, p. 380.
[32] A. Lamkanfi, J. Pérez, and S. Demeyer, "The eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 203–206.
[33] G. Grano, A. Di Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella, "Android apps and user feedback: A dataset for software evolution and quality improvement," in *Proc. 2nd ACM SIGSOFT Int. Workshop App Market Anal.*, Sep. 2017, pp. 8–11.
[34] S. Leone, "Rotten tomatoes movies and critic reviews dataset," Tech. Rep., 2020.
[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[36] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," 2019, *arXiv:1908.10084*.
[37] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. Conf. Empirical Methods Natural Lang. Process., 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, 2019, pp. 3982–3992.
[38] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1877–1901.
[39] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, Feb. 2020.
[40] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRa: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Represent.*, 2021.
[41] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," 2023, *arXiv:2305.14314*.



EYÜP HALIT YILMAZ received the B.Sc. degree in electrical and electronics engineering from Middle East Technical University, Ankara, Türkiye, in 2019, where he is currently pursuing the M.S. degree with the Department of Computer Engineering. His research interests include natural language processing, software applications of AI, large language models, and conversational systems.

İSMAIL HAKKI TOROSLU received the B.S. degree in computer engineering from Middle East Technical University (METU), Ankara, in 1987, the M.S. degree in computer engineering from Bilkent University, Ankara, in 1989, and the Ph.D. degree from the Department of Electrical Engineering and Computer Science, Northwestern University, IL, USA, in 1993. He was a Visiting Professor at the Department of Computer Science, University of Central Florida, from 2000 to 2002. He has been with the Department of Computer Engineering, METU, since 1993. He has published more than 100 technical papers in variety of areas of computer science. His current research interests include data mining, information retrieval, and intelligent data analysis. He has also received IBM Faculty Award, in 2010.

ÖMER KÖKSAL received the B.S. and M.S. degrees from Middle East Technical University and Ph.D. degree from Wageningen University and Research, The Netherlands, in 2018. He has more than 25 years of experience in the industry, he worked as a Software and Research Team Leader and the Project Manager. He has been an Assistant Professor with Department of Data Science and Artificial Intelligence, University of Doha for Science and Technology, Qatar, since 2023. He led software development projects for avionics simulation, command and control, and unmanned control stations. He has also led many research projects based on NLP. He has written numerous academic research articles, technical reports, and book chapters in these domains. His research interests include natural language processing, deep learning, the Internet of Things, and software engineering.