2024 5th International Conference on Artificial Intelligence and Computer Engineering (ICAICE)

A Comparative Study of Bug Triage Representation and Classification Approaches from Canonical to Large Language Models

2024 5th International Conference on Artificial Intelligence and Computer Engineering (ICAICE) | 979-8-3315-2891-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICAICE63571.2024.10864116

Fabiano Tavares da Silva
Sidia R&D Institute
Manaus, Brazil
fabiano.tavares@sidia.com

Felipe Rocha de Araújo*
Sidia R&D Institute
Manaus, Brazil
felipe.araujo@sidia.com

Erick Costa Bezerra
Sidia R&D Institute
Manaus, Brazil
erick.bezerra@sidia.com

Abstract—**Bug triage is the task of assigning newly reported bugs to the proper developers or team for resolution. This is a critical point in software maintenance as it directly influences the time and correct allocation that impact the efficiency and effectiveness of the software process. In a global aspect, the number of teams/developers is extensive, which brings a challenge for bug triage. Traditional approaches to assign bugs struggle with the complexity of the problem. This paper proposes a comparative assessment for different text representation combined with text classification approaches for automated bug report triage by incorporating Large Language Models (LLMs) into the classification pipeline to surpass the limitations of canonical methods. Traditional classification methods were compared with LLM-enhanced models across accuracy metric. The results demonstrate an improvement in triage accuracy when utilizing the fine-tuned LLM, highlighting their potential to provide developer-appropriate bug assignments.**

Keywords*—*bug report triage; LLM; S-BERT

I. Introduction

In the software development life cycle, efficient issue management is essential. One key aspect of this is bug triage, which involves prioritizing and assigning bugs to appropriate developers for resolution [1], [2]. Software systems can have multiple modules or components and can experience a large number of bugs during their development, making it challenging to assign the correct developer to fix it [3]. Moreover, as the number of bugs increases and their requirements vary across skill sets, identifying the right developers or teams with the necessary expertise becomes increasingly difficult [4]. Furthermore, the main information is often encapsulated within the bug report text, requiring an understanding beyond what is explicitly described, adding another layer of complexity to the process.

In this context, a significant branch of AI that can be applied to text content is Natural Language Processing (NLP). NLP focuses on the interaction between computers and humans through natural language, aiming to enable computers to understand, interpret, and generate human language in a meaningful and useful way [5]. It encompasses a wide range of tasks, including text classification, sentiment analysis, and machine translation [6]. Recent advances continue to explore automated approaches using text classification models to streamline bug report triage [1], [7], [8].

A crucial element in enhancing NLP models' performance is the effective text representation, which involves converting text data into a format that can be processed by machine learning algorithms [9]. Word embeddings, a technique in NLP, represent words as dense vectors in a continuous vector space, capturing semantic relationships between words [10]. These embeddings have been improving various NLP tasks providing a richer representation of text data [11].

In [12] a taxonomy to clarify the behavior of feature representation in word embeddings is proposed, with a particular focus on distinguishing between context-independent and context-dependent embeddings. Context-independent embeddings, often referred to as "classical" models, are generated using shallow neural networks that learn fixed representations for words without accounting for their surrounding context. Examples models include Word2Vec [10], GloVe [11], and FastText [13]. In contrast, context-dependent embeddings are produced by Deep Neural Networks (DNN) that extend beyond individual words to capture the meaning of entire sentences, generating dense vectors that reflect the semantic meaning of words in relation to their context. In recent years, DNN like Large Language Models (LLMs) introduced by Vaswani et al. [14], such as Bidirectional Encoder Representations from Transformers (BERT), have proven to be highly effective in generating high-quality sentence embeddings. These embeddings facilitate more accurate and context-aware NLP applications [15].

Based on this context, this study presents a comparative analysis of different text representation (context-independent and context-dependent embeddings) methods and classification techniques applied to the problem of bug report triage, aiming to assign bug reports to the most appropriate development teams across three setups.

* The author is no longer a member of the team and can be reached at felipearaujo@ufpa.br

This study is organized as follows: Section II reviews literature about bug triaging text representation and machine learning. Section III details the methodological approach. Section IV report experiments and setup. Section V describes the results, findings, and discusses the validity of our study compared to previous works. Finally, Section VI contains the conclusions and future work.

II. Related works

In this section, we present studies that evaluated bug triage problems from the aspect of textual representation and textual classification.

Early approaches treated bug triage as an information retrieval or classification problem using traditional machine learning techniques on textual data. For instance, Fanoos et al. [16] conducted a comparative study demonstrating that canonical classifiers can perform effectively.

Mani et al. [17] introduced DeepTriage, employing a deep bidirectional recurrent neural network with attention mechanisms to capture semantic relationships in bug reports. Their model outperformed traditional bag-of-words approaches, emphasizing the importance of rich textual information.

Advancements in NLP and LLMs have also improved text understanding tasks. [9] reviewed recent developments, highlighting how deep learning and pre-training have enhanced semantic analysis and comprehension systems. While LLMs offer improved performance, [18] questioned whether they provide substantial advantages over classical embedding techniques, especially when considering computational resources.

A notable trend in bug triaging is the application of transformer-based models, such as BERT and its variants, which have demonstrated superior performance in text classification tasks. Dipongkor and Moran [19] explored the effectiveness of these techniques, revealing that models like DeBERTa significantly outperform traditional methods in assigning bugs to developers and components. They noted that traditional methods like Term Frequency-Inverse Document Frequency (TF-IDF) based SVMs still performed well in certain contexts, suggesting that transformers might struggle when textual similarity between classes is high. Similarly, Mohanty [20] introduced DEFTri, underscoring the potential of contextual representations in improving classification accuracy.

Meanwhile, Zaidi et al. [7] proposed a bug triage system based on BERT, which not only achieved higher top-k accuracy but also effectively addressed the challenge of incorporating new developers without the need for retraining from scratch.

Moreover, the integration of additional contextual information, such as static analysis and bug reports, has been shown to enhance source code representations for deep learning applications [21]. This approach not only improves performance in software engineering tasks but also emphasizes the importance of data quality and characterization in machine learning, as discussed by Seedat et al. [22] in their TRIAGE framework for regression tasks. The trend towards hybrid models that combine retrieval and classification techniques is also noteworthy, as demonstrated by Meng et al. [23], who proposed a transformer-based system that balances efficiency and accuracy in duplicate bug report detection.

Etemadi et al. [24] proposed a scheduling-driven approach for task assignment, improving upon a state-of-the-art solution. Their approach involved exploring more regions of the search space. Experiments showed that the scheduling-driven approach outperformed the previous method in 71% and 74% of JDT and Platform case studies, respectively.

Zaidi et al. [7] provides an approach uses BERT to create a bug triage system, making it easier to add new developers to an existing project. The model is fine-tuned with manual triage history and tested on large open-source datasets. Results show that this system outperforms other word-embedding methods for bug triage without needing to retrain the entire dataset.

Wei Lu et al. [25] investigated various training strategies for LLMs and discovered that merging fine-tuned models can result in emergent capabilities that surpass the performance of individual models. This finding suggests potential advantages for adapting LLMs to bug triaging tasks, though further research is needed to explore its practical implementation.

In the context of text classification in domains with short and noisy texts, [26] conducted a comparative study from bag-of-words to transformers for classifying healthcare discussions on social media. They discovered that while models like Mistral and BERT were optimal for classifying such texts, traditional word embedding techniques remained viable alternatives with accurate data pre-processing. This finding is relevant to bug triage, as bug reports often contain short, unstructured, and noisy textual data.

Additionally, high textual similarity between classes can hamper model performance, indicating a need for methods that distinguish subtle differences in bug reports. Furthermore, the Hierarchical Attention Network (HAN) proposed by Zhang et al. [27] emphasizes the importance of hierarchical structures in processing bug reports to improve the prioritization and classification for issues.

Another gaps exists in the integration of advanced LLMs into bug triage systems. While transformer-based models have demonstrated potential, their application in bug triage remains underexplored. Dipongkor et al. [19] suggested that the distinct characteristics of various techniques enable them to capture different patterns in bug reports, indicating that ensemble methods combining traditional and deep learning approaches may improve performance. Building on this insight, our approach aims to examine the impact of feature extraction on text classification, ranging from classical methods to LLMs.

III. Methodology

The experiments were conducted in three main phases: III-A Dataset Preprocessing; III-B Feature Extraction, where different methods were employed to generate diverse text representations across multiple scenarios; and IV Setup and Model Training, where three scenarios were combined with the previous steps to train models for automated bug triage. Figure 1 outlines this key steps involved in the proposed automated

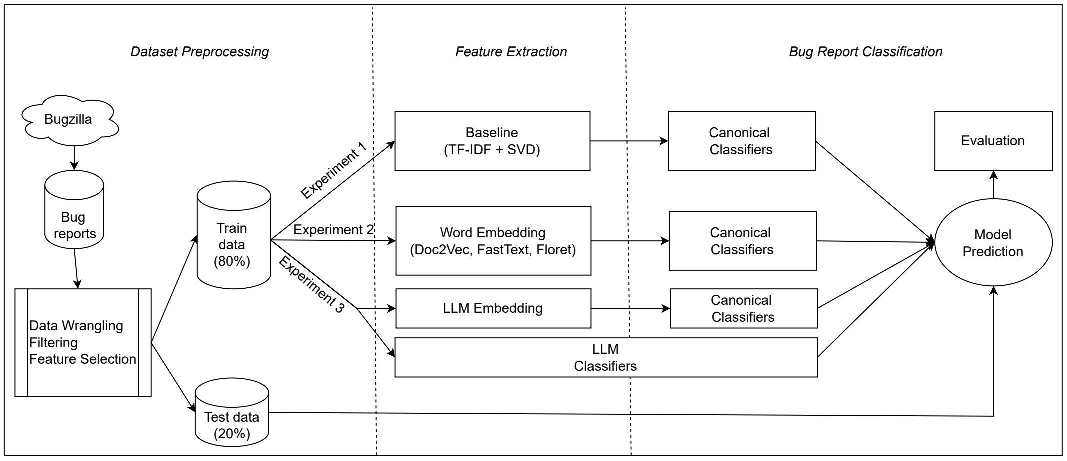

Fig. 1. Overview of the text classification experiments for bug report triage on Bugzilla dataset

bug triaging experiments, which are explained in detail in the subsequent subsections.

A. Dataset Preprocessing

For our research, the bug data was collected from Bugzilla tool on Mozilla[1] projects website. This data is a rich history of bug reports and resolutions related with open-source software repositories.

A historical bug report dataset was constructed using data extracted from Bugzilla, comprising ten years of reported bugs from 2014 to 2024. The dataset includes not only the final status of each bug report but also all modifications that occurred throughout its lifecycle. These reports contain textual information and categorical attributes. Table I provides a detailed overview of the features in the complete dataset.

After extracting data from the Bugzilla tool, we performed a preliminary analysis and began the pre-processing phase, which involved the following steps:

1) Data Wrangling: Raw data were cleaned and unified into an accessible format to ensure data quality and readiness for analysis (Table representation).

2) Data Filtering: Using tabular data, we select relevant parts of instances to focus on data that significantly contributes to the classification task. The duplicate attribute was removed due to 97% of missing data. Any bug instance without ”Assigned_to” or marked as ”nobody” was also removed. About the number of tokens, textual data were eliminated whether they contained less than 3 tokens or exceeded 255 tokens, determined by outlier detection methods in [28]. For the purpose of correct classification, filtering considered static bugs with a resolution FIXED and the status RESOLVED or VERIFIED.

3) Feature selection: The most relevant features were identified, based on their relevance to the target variable and potential predictive power. The features shown in Table I are

TABLE I
BUG REPORT'S ATTRIBUTES DESCRIPTION

Attribute		Description
Textual	Summary	A brief overview or description of the bug encountered.
	Description	A detailed explanation of the issue, steps to reproduce it, and any additional information relevant to understanding the problem.
Categorical	Component	Identifies the specific area within the product where the bug occurred.
	Priority	Specifies how critical or important the bug is, ranging from high to low priority.
	Version	Indicates the software version where the bug was detected.
	Product	Defines which product the bug report belongs to.
	Op system	Specifies the operating system on which the bug was found.
	Severity	Measures the impact of the bug on the overall functionality and user experience, classified as critical, high, medium, or low severity.
	Platform	Identifies the platform where the bug is encountered (e.g., web, mobile, desktop).
Target	Assigned_to	Specifies the team member or engineer responsible for addressing and resolving the reported issue.

selected using the 95% cumulative sum of feature importance using the Random Forest approach [29].

Resulting in 203,025 bugs reports distributed according to Table I. The “Assigned_to” attribute as highlighted in [30], is subject to change in 29% to 66% throughout the bug's lifecycle.

An analysis of the ”assigned_to” distribution revealed a imbalance data, reflecting the lack of clear distinction in assignment developer or team, which suggests a high level of granularity in certain instances of data. To build a feasible training dataset, we established a threshold of a minimum of

[1]Bugzilla, http://www.bugzilla.org/

37 bugs per target class based on interquartile range outlier detection method [28].

As showed in Figure 2, the data distribution indicates that, over time (horizontal axis), the target attribute (developer or team) becomes obsolete or is replaced, presenting a significant challenge of overlapping interdependence between certain classes. While the distribution of the top classes remains stable across years (as shown in (B)), the same does not apply to the tail classes, as seen in (A). This distribution was preserved in the experiments to accurately reflect real-world software development environments.

The dataset was divided into two parts: a training (80%) of the data and a testing (20%). This division was based on the chronological distribution of the data, ensuring that the testing set included the most recent bug reports for each ”assigned to” category. This approach was designed to simulate real-world.

B. Feature Extraction

In the dataset obtained from Bugzilla, detailed in Table I, the attributes predominantly consist of textual and categorical data. Each attribute type go through feature extraction and conversion into numeric vector representations. In machine learning, this process is crucial for machines to understand and classify data in a manner analogous to human understanding [31].

Categorical data are processed through a separate pipeline. Categories containing fewer than twenty unique elements are encoded using One-Hot Encoding, transforming each categorical value into a binary vector, ensuring minimal loss of information due to encoding [32]. The categorical attributes were consistently represented in this format throughout the experiments, except when an LLM-based classification was introduced, at which point it was integrated with the textual attributes.

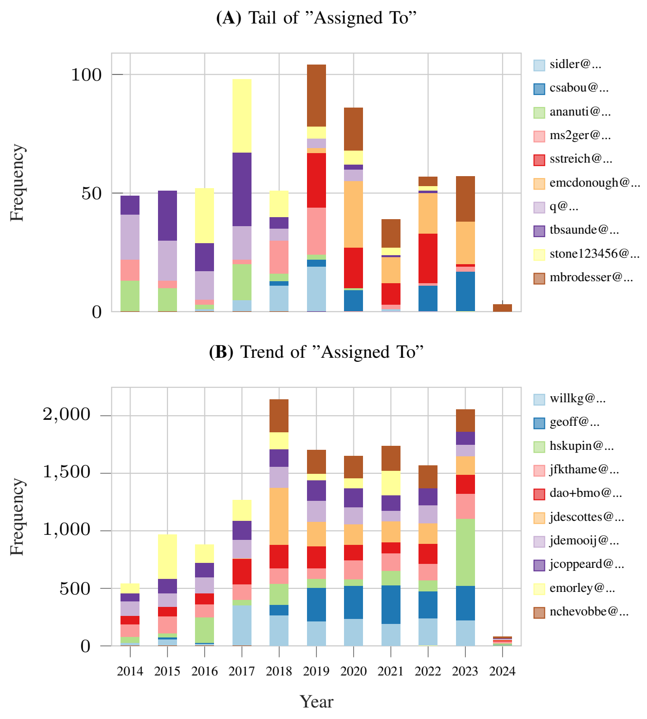

Fig. 2. Distribution of number of bug report over the year by ”assigned to” to top 10 classes

Regarding the textual attributes, titles and descriptions, are processed through a pipeline designed to represent them as numerical vectors. As suggest by [33], this modification is essential for the classification process, as expanding the input representation space enhances the likelihood of data separability. Consequently, as shown in Figure 1, our research employed three different textual representations, detailed with specific methods below:

- **Baseline:** This conversion involves an initial transformation using TF-IDF to quantify the importance of each token within the text corpus. TF-IDF is a widely used feature extraction method within the Bag-of-Words (BoW) framework for text representation in natural language processing tasks. TF-IDF assigns weights to terms based on their importance across documents, enhancing the representation of significant words in a corpus [34]. However, TF-IDF representations often result in high-dimensional and sparse matrices, posing computational challenges for machine learning algorithms. To address this, Singular Value Decomposition (SVD) is applied for dimensionality reduction similar was applied by [33]. This process not only reduces computational complexity but also enhances the performance of text classification task by capturing latent relationships among terms [35].
- **Word Embeddings:** Classic word embeddings have significantly advanced NLP by providing dense vector representations of words that capture semantic and syntactic relationships [10]. Unlike traditional BoW, which represent text as high-dimensional sparse vectors without accounting for word order or context, embeddings enable models to understand linguistic nuances, thereby enhancing performance in tasks such as text classification. We used prominent embedding techniques in our study including Doc2Vec [36], FastText [13], and Floret [37]. Doc2Vec extends the Word2Vec [10] framework to produce vector representations for entire documents by considering the context in which words appear, facilitating document-level semantic analysis. FastText builds upon Word2Vec by incorporating subword information, representing words as character n-grams, which allows it to generate embeddings for rare or misspelled words. Floret further enhances this approach by providing efficient embeddings optimized for resource-constrained environments, making it suitable for applications where memory and computational power are limited [37].
- **Large Language Models:** LLMs brings contextualized embeddings with an advanced text representation by capturing the dynamic meanings of words based on their context within sentences [15]. Unlike traditional static embeddings, which assign a single vector to a word re-

gardless of context, contextualized embeddings generate different representations for the same word depending on its surrounding words. During training, LLMs extract new features from input data through embedding, which is crucial for fine-tuning pre-trained models used for tokenization and representing data [38]. BERT is a prime example of an LLM that produces contextualized embeddings [39]. BERT utilizes a transformer architecture that processes text bidirectionally, meaning it considers both left and right contexts simultaneously. It is pre-trained on large corpora using masked language modeling and next sentence prediction tasks, enabling it to learn deep semantic and syntactic representations of language. For sentence embedding, BERT generates fixed-length vector representations that encapsulate the meaning of entire sentences by averaging or pooling token embeddings from its final layers [15]. These sentence embeddings have been effectively used in task of text classification that enhances the model's understanding of language.

IV. Experimental setup

The methodology comprises three experiments, as showed in Figure 1. The experiments investigate both traditional machine learning algorithms and modern approaches with LLM, providing a comprehensive exploration of bug report triage methods.

1) Experiment 1: TF-IDF+SVD with Canonical Classifiers: In the first experiment, we employed 10 traditional classification algorithms to address the bug report triage task. This experiment serves as a baseline for future comparisons, allowing direct assessment of performance. The training dataset of bug reports was processed through text representation via feature extraction using the TF-IDF method, followed by SVD feature reduction. The integration of TF-IDF and SVD provides an effective text representation while minimizing redundancy and noise in the feature set [33]. The TF-IDF was configured utilizing L2 normalization, ensuring the sum of squared vector elements is equal to 1 and applying a smooth IDF weight by incorporating it into document frequencies to avoid division by zero [40].

For the title and description attributes, this approach produced vector representations consisting of 20 and 100 features, respectively. Categorical data were encoded using One-Hot Encoding. The textual feature vectors, categorical vectors, and target labels were then concatenated into a single vector and used as input to a series of canonical classifiers. To mitigate bias in the training set and identify the optimal model, stratified 10-fold cross-validation was applied to the training data during the training process.

This experiment was conducted using several classifiers, including Ada Boost (ADA), CatBoost (CB), Decision Tree (DT), Extreme Gradient Boosting (XGB), K-Nearest Neighbors (KNN), Light Gradient Boosting Machine (LGBM), Linear Discriminant Analysis (LDA), Logistic Regression (LR), Naive Bayes (NB), and Quadratic Discriminant Analysis (QDA). The models were implemented using PyCaret (version 3) and Scikit-learn (version 1.5), both well-established machine learning frameworks. PyCaret provided a streamlined pipeline for data preprocessing, model training, and performance evaluation, facilitating efficient comparison of multiple models, while Scikit-learn enabled fine-tuning of the classification algorithms. All algorithms were run with default hyperparameter settings.

2) Experiment 2: Word Embeddings with Canonical Classifiers: The TF-IDF+SVD was replaced by word embedding techniques for representing textual attributes. Rather than relying on traditional vectorization methods, we employed Doc2Vec, FastText, and Floret, each providing distinct methods for capturing semantic meaning in text. Doc2Vec, implemented using the Gensim library[2], generated document-level embeddings, while FastText and Floret, accessed through their original GitHub implementations[3], incorporated sub-word information, making them particularly effective for handling domain-specific language [13].

In contrast to the first phase, this experiment involved an additional training step for the word embedding models using the same dataset intended for classifier training. Word2Vec, FastText, and Floret use neural networks with a single hidden layer to learn word embeddings. During this process, both a context window and vector size were specified, with a window size of 5 and a vector size of 300 applied to all models. Once the embedding model was trained, it was used to generate vector representations for each textual attribute of a preprocessed and tokenized bug report.

The subsequent steps, including categorical representation, vector concatenation, and input into the list of canonical classifiers for evaluation, followed the same steps as in the first experiment. This experiment aimed to assess whether these embedding techniques could outperform the baseline in terms of classification accuracy.

3) Experiment 3: LLMs-based: In the third experiment, illustrated in Figure 1, both the textual representation and the classifier were modified. Pre-trained LLMs from the BERT family were employed as LLM embedding for text representation. These models were applied in two forms: as pre-trained models and as fine-tuned versions for feature extraction. Similar to the second experiment, this approach required training a model to generate dense vector representations for sentences. When using a pre-trained model, the model was applied directly in a single forward pass to generate the last hidden layer, which contains the embedding representing the entire processed sentence. LaBSE and all-MPNet-v2 were used as pre-trained models.

For fine-tuning, following a similar procedure as in the second experiment, a pre-trained LLM was further trained on the dataset to generate sentence embeddings. The BERT-base-uncased[4] model was employed for this purpose, and the

[2]https://radimrehurek.com/gensim/

[3]https://github.com/explosion/floret

[4]https://huggingface.co/google-bert/bert-base-uncased

TABLE II
RESULTS FROM EVALUATING THE MAIN EXPERIMENTS COMBINING TEXT REPRESENTATION AND TEXT CLASSIFICATION METHODS.

	Feature Extraction	Metrics\	ADA	CB	DT	XGB	KNN	LGBM	LDA	LR	NB	QDA	LLM
Experiments 1: Baseline	**TF-IDF + SDV**	**Acc@1**	1,61%	**24,73%**	19,19%	4,41%	7,99%	2,18%	7,87%	12,10%	3,03%	3,74%	n\a
		F-1	0,15%	22,48%	18,39%	2,99%	7,84%	2,16%	7,33%	10,78%	2,61%	3,34%	n\a
		Acc@5	6,45%	48,78%	19,75%	8,52%	16,79%	3,07%	19,98%	24,80%	6,99%	8,15%	n\a
Experiments 2: Context-independent	**Doc2Vec**	**Acc@1**	1,61%	15,77%	**17,09%**	4,34%	1,35%	0,62%	2,97%	3,17%	1,40%	1,14%	n\a
		F-1	0,15%	13,76%	16,28%	1,38%	1,06%	0,03%	2,16%	2,66%	1,07%	0,55%	n\a
		Acc@5	6,45%	35,74%	17,61%	7,63%	4,25%	1,30%	8,70%	9,66%	4,41%	2,95%	n\a
	FastText	**Acc@1**	1,61%	22,65%	18,74%	4,60%	9,63%	1,05%	13,45%	12,63%	3,95%	2,37%	n\a
		F-1	0,72%	21,03%	17,99%	1,76%	9,67%	0,10%	14,12%	12,08%	3,68%	0,78%	n\a
		Acc@5	4,52%	45,98%	19,25%	7,26%	21,13%	1,72%	27,48%	26,10%	8,99%	3,36%	n\a
	Floret	**Acc@1**	1,61%	**27,09%**	20,53%	7,47%	17,16%	0,76%	18,32%	16,91%	6,63%	2,65%	n\a
		F-1	0,15%	25,68%	19,77%	4,00%	16,86%	0,10%	19,18%	16,41%	6,38%	1,08%	n\a
		Acc@5	6,45%	54,09%	21,06%	11,96%	34,51%	1,44%	39,45%	36,01%	15,14%	3,64%	n\a
Experiments 3: Context-dependent	**all-mpnet-base-v2**	**Acc@1**	1,61%	28,38%	19,91%	6,04%	21,49%	0,65%	17,73%	16,05%	7,83%	1,41%	28,55%
		F-1	0,72%	26,09%	19,18%	3,09%	21,07%	0,21%	18,34%	15,45%	7,39%	0,81%	23,98%
		Acc@5	4,52%	55,22%	20,40%	9,61%	41,46%	1,35%	38,31%	33,01%	16,92%	2,45%	56,08%
	LaBSE	**Acc@1**	1,61%	28,98%	20,04%	2,72%	18,18%	2,67%	17,73%	15,90%	8,46%	0,43%	29,80%
		F-1	0,72%	26,69%	19,23%	1,88%	17,95%	1,57%	18,81%	15,55%	8,34%	0,43%	25,27%
		Acc@5	4,52%	54,68%	20,55%	7,90%	35,16%	3,45%	35,13%	33,42%	17,93%	1,11%	59,64%
	BERT+fine-tunning	**Acc@1**	1,62%	**30,48%**	20,31%	7,34%	18,81%	2,57%	22,36%	20,33%	10,42%	0,32%	29,05%
		F-1	0,07%	28,20%	19,56%	5,67%	18,50%	2,41%	23,53%	19,93%	10,12%	0,35%	24,44%
		Acc@5	4,53%	57,46%	20,81%	10,23%	37,24%	3,89%	43,10%	41,03%	21,53%	1,00%	56.64%

Transformer-based Denoising AutoEncoder (TSDAE) algorithm [41] was applied to produce the fine-tuned model for sentences. Fine-tuning allowed the models to learn domain-specific language and context, which general pre-trained embeddings may not fully capture [25]. Aside from changes to the textual representation, the subsequent steps for the classifiers and evaluation remained consistent across all before experiments.

Additionally, in the third experiment, a final experiment was conducted where the LLM classifier itself, rather than a traditional classifiers, was employed for both textual representation and classification. We used the BERT family for text classification tasks. The input text is tokenized and incorporates special tokens: [CLS] at the beginning and [SEP] to denote the end or separate features—both textual and categorical—differently from previous experiments. The [CLS] token provides a dense vector representation of the input, while [SEP] helps manage multi-sentence inputs. These tokens, along with the rest of the text, pass through DNN layers, where self-attention creates contextual embeddings. The final state of the [CLS] token is sent to a head classification layer, which maps it to the target class. The model is fine-tuned with a task-specific classification head and optimized using AdamW with learning rate of 2e-5. The model is trained by adjusting its parameters to improve classification accuracy. Table III provides detailed information on the characteristics of the models used.

In both cases, the Hugging Face Transformers and Sentence Transformers[5] libraries were used to implement and apply these models. The primary aim of this experiment was to evaluate whether LLMs, either pre-trained or fine-tuned, could better handle complex and ambiguous bug descriptions, potentially improving upon the results of the previous experiments.

All experiments were conducted on a server with 48GB of RAM, a 40GB A100-40C GPU, and a CUDA 12.2 driver, which provided sufficient resources for inference tasks. However, for training, we had to utilize half precision (fp16) and limit the model size to a maximum of half a billion parameters due to hardware constraints. The server also featured an Intel(R) Xeon(R) Gold 6552 CPU running at 2.10 GHz with 8 cores.

[5]https://huggingface.co/ and https://sbert.net/

TABLE III
COMPARISON OF LLM MODELS.

Feature	LaBSE	all-mpnet-base-v2	BERT-base-uncased
Model architecture	BERT	MPNet	BERT
Number of layers	12	12	12
Hidden size	768	768	768
Attention heads	12	12	12
Feed-forward hidden size	3072	3072	3072
Max sequence length	512	384	512
Total parameters	340M	110M	110M
Vocabulary size	119,547	30,000	30,000
Optimizer	AdamW	AdamW	AdamW

V. EVALUATION AND RESULTS

In this section, we analyze the performance of various models across the three experimental scenarios described in Section IV.

To evaluate the models' performance, we employed a holdout technique, using 20% of the dataset—unseen by any model during training—as the test set. This dataset was used consistently across all three experiments. The model prediction in Bug Report Classification is depicted in Figure 1. To ensure unbiased evaluation, all bug reports underwent the same preprocessing steps applied during the training phase, followed by each feature extraction method outlined in Section III-B.

We employed several key evaluation metrics, including accuracy, F1-score, and top-5 accuracy, chosen for their ability to provide a holistic assessment of the models' effectiveness in classifying bug reports across multiple labels. These metrics are commonly used in bug triage problems [16]. Accuracy measures the percentage of correctly classified instances out of the total number of instances, while top-5 accuracy evaluates whether the correct label is among the top five predictions. Precision quantifies the proportion of true positives among all predicted positives, and recall measures the proportion of true positives among all actual positives. The F1-score is the harmonic mean of precision and recall, provides a balanced evaluation metric.

As shown in Table II, the CatBoost Classifier demonstrated the best performance, achieving the highest accuracy of 24% and an F1-score of 21%. Other classifiers, such as Decision Tree and K-Neighbors, displayed moderate performance (Experiment 1), while AdaBoost and Naive Bayes struggled, with significantly lower accuracy and F1-scores. This experiment established a baseline.

In the second experiment, word embeddings (Doc2Vec, FastText, and Floret) were evaluated. Among these, Floret consistently outperformed the others with accuracy of 27,09% and an F1-score of 25,68% (Experiment 2), particularly when used with CatBoost and Decision Tree classifiers. This result highlights Floret's ability to capture subword information, which is crucial for domain-specific tasks such as bug triage.

The third experiment introduced LLM-based embeddings, further enhancing performance. BERT fine-tuning emerged as the top-performing LLM, achieving the highest accuracy (30.5%) and F1-score (28%) across multiple classifiers. While models like LaBSE and all-MPNet-v2 produced competitive results. These findings demonstrate that LLM embeddings significantly improve classification performance compared to traditional word embeddings. Fine-tuning pre-trained models provided a clear advantage over using embeddings directly from LLMs, with a boost in performance, particularly to leading to more accurate team assignments.

The overall accuracy results were relatively low, consistent with the findings of Fanoos et al. [16], which suggest that accuracy tends to decline as the number of classes increases, as observed in this study with 626 classes. Figure 3 illustrates the gradual execution of Experiment 3, increase the number of class for bug triage, where the LLM (LaBSE) was used only for embedding with the CatBoost classifier and the LLM (LaBSE) being used for both embedding and classification. The results indicate that the LLM classifier accuracy performed well in lower-dimensional spaces but gradually lost effectiveness in higher-dimensional settings. Additionally, in the context of a large number of classes and an imbalanced dataset, tree-based classifiers demonstrated superior performance, aligning with findings in recent literature [42] [43].

VI. Conclusion

In this study, we implemented a bug triage process using a common dataset for model training, combining supervised classification techniques and text embeddings ranging from traditional methods to state-of-the-art (LLMs). By predicting on test data, we analyzed and evaluated the performance of various text representation and classification methods in terms of accuracy and efficiency for bug assignment.

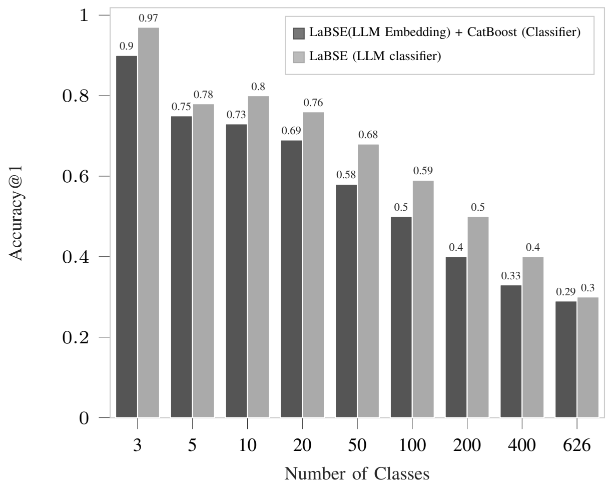

Fig. 3. Accuracy comparison varying the number of classes.

The three experimental scenarios together offer a comprehensive evaluation of different approaches to addressing the bug triage problem. The progression from traditional vectorization techniques to fine-tuned LLMs highlights the critical role of advanced textual representation in improving classification accuracy and model performance for real-world applications.

Overall, the results across the three scenarios demonstrate a clear improvement in performance as more advanced text representation techniques were introduced. The baseline models, which employed TF-IDF and traditional classifiers, provided a foundation for comparison. However, the shift to more sophisticated word embeddings and, ultimately, LLM embeddings, led to significant gains in classification accuracy. Fine-tuning pre-trained models consistently yielded the best results, proving to be the most effective approach across all stages of the experiments.

These findings emphasize the importance of advanced text representation techniques, particularly for tasks involving complex textual data like bug triage. Fine-tuning pre-trained models such as BERT base for domain-specific tasks has the potential to enhance performance, making it a strategy for optimizing bug triage systems in real-world software development environments.

Future research should address the GPU limitations. Training larger LLMs requires processing capabilities beyond those used in these initial experiments. Nonetheless, the observed trend of improved performance with larger models suggests promising potential. Additionally, further exploration of hyperparameter optimization is necessary, as the default settings used in this study may not have fully optimized classifier performance. Finally, the relatively classification performance of the LLM-based classifier suggests the need to explore strategies commonly used in tree-based models, such as hierarchical structure and ensemble learning, to improve classification outcomes.

Acknowledgment

This work is the result of the RD project *Engenharia de Software e Ciência de Dados aplicados ao Desenvolvimento de Sistemas*, in partnership with Samsung Eletrônica da Amazônia Ltda., under the stimulus of the Brazilian Federal Law No. 8.387/91. Its disclosure and publicity are under provisions of Article 39 of Decree No. 10.521/2020.

References

[1] M. Alenezi, S. Banitaan, and M. Zarour, "Using categorical features in mining bug tracking systems to assign bug reports," *International Journal of Software Engineering & Applications*, vol. 9, 03 2018.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*. ACM, 2006, pp. 361–370.

[3] W. Zhang, "Efficient bug triage for industrial environments," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 727–735.

[4] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. ACM, 2009, pp. 111–120.

[5] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed., 2024, online manuscript released August 20, 2024.

[6] Y. Goldberg, *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[7] S. F. A. Zaidi, H. Woo, and C.-G. Lee, "Toward an effective bug triage system using transformers to add new developers," *Journal of Sensors*, vol. 2022, no. 1, p. 4347004, 2022.

[8] A. Patil and A. Jadon, "Auto-labelling of bug report using natural language processing," in *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*, 2023, pp. 1–7.

[9] M. Basha, V. Selvaraj, J. Jayashankari, A. Alawadi, and P. Durdona, "Advancements in natural language processing for text understanding," *E3S Web of Conferences*, vol. 399, 07 2023.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[12] C. Wang, P. Nulty, and D. Lillis, "A comparative study on word embeddings in deep learning for text classification," in *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, ser. NLPIR '20. New York, NY, USA: Association for Computing Machinery, 2021, p. 37–46.

[13] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[14] A. Vaswani, N. Shazeer, and N. e. a. Parmar, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[15] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[16] M. Fanoos, A. Hamdy, and K. Nagaty, "Bug triage automation approaches: A comparative study," *International Journal of Open Source Software and Processes*, vol. 13, pp. 1–19, 01 2022.

[17] S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," in *Proceedings of the ACM India joint international conference on data science and management of data*, 2019, pp. 171–179.

[18] M. Freestone and S. K. K. Santu, "Word embeddings revisited: Do llms offer something new?" 2024.

[19] A. K. Dipongkor and K. Moran, "A comparative study of transformer-based neural text representation techniques on bug triaging," 2023.

[20] I. Mohanty, "Deftri: A few-shot label fused contextual representation learning for product defect triage in e-commerce," 2023.

[21] X. Guan and C. Treude, "Enhancing source code representations for deep learning with static analysis," 2024.

[22] N. Seedat, J. Crabbé, Z. Qian, and M. van der Schaar, "Triage: Characterizing and auditing training data for improved regression," 2023.

[23] Q. Meng, X. Zhang, G. Ramackers, and V. Joost, "Combining retrieval and classification: Balancing efficiency and accuracy in duplicate bug report detection," 2024.

[24] V. Etemadi, O. Bushehrian, R. Akbari, and G. Robles, "A scheduling-driven approach to efficiently assign bug fixing tasks to developers," *Journal of Systems and Software*, vol. 178, p. 110967, 2021.

[25] W. Lu, R. K. Luu, and M. J. Buehler, "Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities," 2024.

[26] E. De Santis, A. Martino, F. Ronci, and A. Rizzi, "From bag-of-words to transformers: A comparative study for text classification in healthcare discussions in social media," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–15, 2024.

[27] A. Yadav and S. S. Rathore, "A hierarchical attention networks based model for bug report prioritization," in *Proceedings of the 17th Innovations in Software Engineering Conference*, ser. ISEC '24. New York, NY, USA: Association for Computing Machinery, 2024.

[28] N. M. R. Suri, M. N. Murty, and G. Athithan, *Outlier detection: techniques and applications*. Springer, 2019.

[29] H. Elghazel and A. Aussem, "Unsupervised feature selection with ensemble learning," *Machine Learning*, vol. 98, pp. 157–180, 2015.

[30] A. Lamkanfi, J. Perez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information," in *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories, May 18-–19, 2013. San Francisco, California, USA*, 2013.

[31] U. Naseem, I. Razzak, S. K. Khan, and M. Prasad, "A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models," *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, vol. 20, no. 5, jun 2021.

[32] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image and Vision Computing*, vol. 75, p. 21–31, Jul. 2018.

[33] A. I. Kadhim, Y.-N. Cheah, I. A. Hieder, and R. A. Ali, "Improving tf-idf with singular value decomposition (svd) for feature extraction on twitter," in *3rd international engineering conference on developments in civil and computer engineering applications*, 2017.

[34] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

[35] M. Berry, "Survey of text mining," 2004.

[36] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.

[37] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, April 2017, pp. 427–431.

[38] J. Dagdelen, A. Dunn, S. Lee, N. Walker, A. S. Rosen, G. Ceder, K. A. Persson, and A. Jain, "Structured information extraction from scientific text with large language models," *Nature Communications*, vol. 15, no. 1, p. 1418, 2024.

[39] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[40] R. Baeza-Yates and B. Ribeiro-Neto, "Modern information retrieval, second penyunt," 2011.

[41] K. Wang, N. Reimers, and I. Gurevych, "Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning," 2021.

[42] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from imbalanced data sets*. Springer, 2018, vol. 10, no. 2018.

[43] J. Tanha, Y. Abdi, N. Samadi, N. Razzaghi, and M. Asadpour, "Boosting methods for multi-class imbalanced data classification: an experimental review," *Journal of Big data*, vol. 7, pp. 1–47, 2020.