Software Quality Journal (2024) 32:985–1005
https://doi.org/10.1007/s11219-024-09675-3

RESEARCH

LLM-BRC: A large language model-based bug report classification framework

Xiaoting Du[1,2] · Zhihao Liu[3] · Chenglong Li[3] · Xiangyue Ma[3] · Yingzhuo Li[1] · Xinyu Wang[1]

Accepted: 23 April 2024 / Published online: 24 May 2024
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2024

Abstract

Deep learning frameworks serve as the cornerstone for constructing robust deep learning systems. However, bugs within these frameworks can have severe consequences, negatively affecting various applications. Accurately classifying and understanding these bugs is essential to ensure framework reliability. By doing so, developers can proactively take appropriate measures to mitigate potential risks associated with specific bug types in both current and future software releases. Despite the significance of bug report classification, existing methods fall short in terms of performance, rendering them impractical for real-world applications. To address this limitation, we propose a bug report classification framework for deep learning frameworks, called LLM–BRC, leveraging OpenAI's latest embedding model, *text-embedding-ada-002*. Our LLM–BRC framework achieves an impressive accuracy range of 92% to 98.75% in bug report classification for three deep learning frameworks: TensorFlow, MXNET, and PaddlePaddle. This represents a substantial improvement of 17.21% to 69.15% compared to existing methods. Furthermore, we conduct a comprehensive investigation into the impact of different bug report components and different models.

Keywords Bug report classification · Deep learning framework · Large-language model

✉ Xiaoting Du
duxiaoting@bupt.edu.cn

Chenglong Li
li_chenglong@buaa.edu.cn

[1] School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing, China

[2] Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China

[3] School of Automation Science and Electrical Engineering, Beihang University, Beijing, China

1 Introduction

Deep learning frameworks play a crucial role in building robust deep learning systems (Zhang et al., 2020). With the rapid advancement of deep learning technology, the demand for deep learning frameworks has experienced exponential growth (Guo et al., 2018). This expansion encompasses the incorporation of new interfaces, the enhancement of functionalities, and the optimization of compatibility with a wide array of hardware devices and underlying drivers. Throughout this evolutionary process, the continuous iteration of code and version updates inevitably introduces bugs into deep learning frameworks (Zhang et al., 2018). Bugs in deep learning frameworks can have a significant and wide-reaching impact on a larger user base compared to specific deep learning models. Particularly in safety- and security-critical domains like autonomous driving (Chen et al., 2015) and healthcare (Cai et al., 2014), the consequences of these bugs can be more severe. Therefore, ensuring the reliability of deep learning frameworks is of utmost importance.

Numerous studies have been conducted to gain insights into the characteristics of bugs in deep learning frameworks and provide assistance in their resolution. For instance, Jia et al. (2021) conducted an analysis of bugs in TensorFlow based on 202 bug fixes. The findings revealed that bugs in TensorFlow can be classified into 6 distinct categories based on symptoms and 11 distinct categories based on root causes. In (Islam et al., 2019), Islam et al. examined five deep learning libraries, namely Caffe (Jia et al., 2014), Keras (Lux & Bertini, 2019), TensorFlow (Girija, 2016), Theano (Team et al., 2016) and Torch (Collobert et al., 2002). They analyzed 2,716 posts from Stack Overflow and 500 bug fix commits from GitHub to identify commonly occurring bug types in deep learning frameworks. According to the classification results, there are five different bug types, including API bugs, Coding bugs, Data bugs, Structural bugs, and Non model structural bugs. In (Du et al., 2022), we conducted a classification of bug reports in TensorFlow, MXNET, and PaddlePaddle based on fault-triggering conditions. Bugs were categorized into Bohrbugs (BOHs) and Mandelbugs (MANs), taking into account the conditions of fault activation and error propagation. Moreover, within the MAN category, bugs were further classified as either non-aging related Mandelbugs (NAMs) or aging-related bugs (ARBs).

However, the bug classification process in the aforementioned studies was all performed manually. As the number of bug reports in deep learning frameworks continues to increase, manually classifying all bug reports becomes impractical. Therefore, the development of bug report classification methods becomes essential. In (Xia et al., 2014), the authors employed the bag-of-words model to represent bug reports and utilized machine learning classifiers to classify them. However, the bag-of-words model neglects the contextual semantic information present in bug reports, resulting in inadequate classification results.

To address this limitation and effectively utilize the semantic information embedded within bug reports, we proposed the DeepSIM method in Du et al. (2021). DeepSIM employed a word2vec semantic model that was trained based on over two million bug reports. However, the effectiveness of DeepSIM is hindered by the constrained size of the training corpus utilized for the semantic model. To address the aforementioned issues, we propose a Large Language Model-based Bug Report Classification framework (LLM–BRC) for deep learning frameworks. Large language models (LLMs), particularly GPT-3 and GPT-4 (Brown et al., 2020; Radford et al., 2018, 2019) have proven transformative in numerous fields and have made remarkable contributions in domains ranging from mathematics (Frieder et al., 2023) and communication (Guo et al., 2023) to even medicine (Nov et al., 2023). In particular, the prowess of LLMs lies in their ability to revolutionize

text processing across diverse tasks, substantially propelling the fields of natural language understanding and generation to new heights (Ray, 2023). One of the core strengths of LLMs is their mastery of language representation through dense vector embeddings. By capturing intricate semantic meaning and contextual information, these embeddings allow for a more nuanced understanding of language and context-aware language processing.

In our framework, we leverage the *text-embedding-ada-002* model, which is the second-generation embedding model announced by OpenAI on December 15, 2022, to represent bug reports and facilitate bug report classification. Based on this model, bug reports can be transformed into embeddings of a dimension size of 1,536. These embedding vectors are then fed into a feed-forward neural network (FFN) for bug report classification. Unlike traditional machine learning classifiers, FFN excels at capturing intricate patterns and dependencies within the data, enabling it to learn highly representative and discriminative features. This allows for enhanced accuracy of bug report classification and the ability to handle high-dimensional input data efficiently. Finally, the effectiveness of LLM–BRC is evaluated on bug reports from three deep learning frameworks.

In summary, this article makes the following main contributions.

1. We present LLM–BRC, a Large Language Model-based Bug Report Classification framework that combines a large language model with a deep learning classifier. Through this method, we achieved accurate classification of bugs in deep learning frameworks, with an accuracy ranging from 92% to 98.75%.
2. We explore the factors influencings classification results, including information from different components of bug reports and types of language models, to further promote the practical application of this method.
3. In order to facilitate bug report classification research, we have open-sourced both the data and the method, which can be accessed at the following webpage: https://sites.google.com/view/llmbp/.

The rest of the paper is organized as follows. Section II presents the proposed approach. Section III provides an overview of the experimental setup. Section IV describes the evaluation and analysis of the results. In section V, we discuss the threats to validity. Section VI presents the related work. Finally, the last section concludes the paper.

2 Our approach

In this section, we propose a bug report classification framework called LLM–BRC. The overall procedure of LLM–BRC is depicted in Fig. 1. As shown in the figure, LLM–BRC comprises three sequential steps: data preparation, LLM-based bug report representation, and bug report classification. In the data preparation phase, we start by extracting information from bug reports in deep learning frameworks' GitHub repositories, using a custom-designed web crawl tool. Next, the preprocessed bug reports are fed into the OpenAI's *text-embedding-ada-002* model, which transforms the natural language text into dense embedding vector representations. These embeddings capture the semantic meaning and contextual information present in the bug reports. Finally, a FFN is constructed and trained using labeled bug reports. The FFN utilizes the learned embeddings to perform the bug report classification task. In the subsequent parts of this section, we provide a detailed explanation of each step of LLM–BRC.

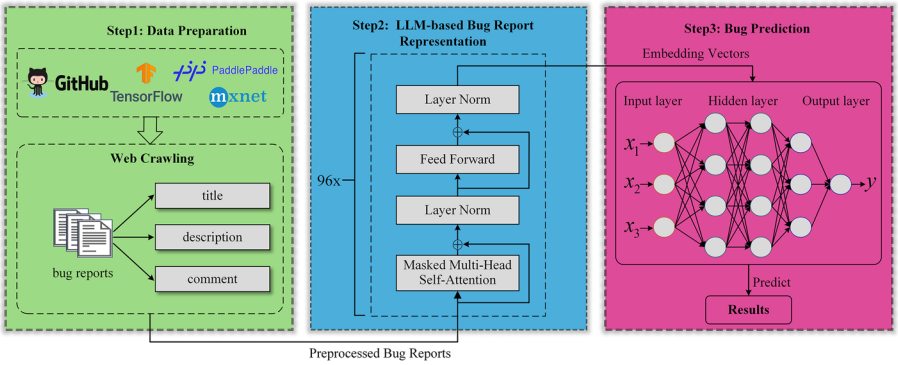

Fig. 1 Detailed structure of LLM-BRC

2.1 Data preparation

We initiate the data preparation process by crawling bug reports based on their Bug-ID from the GitHub repositories of TensorFlow, MXNET, and PaddlePaddle. This crawl phase considers a total of 3,110 bug reports from these three deep learning frameworks, which were previously labeled in our previous work (Du et al., 2022). Since text is the dominant feature contained in bug reports, we collect natural language information including title, description, and comments from each bug report. Among them, the title provides a concise summary of the entire bug report, offering a brief overview of the entire bug report. The description section contains a detailed account of the issue, including observed software anomalies, the software runtime environment, reproduction steps, and other relevant details. Furthermore, the comment section comprises discussions and exchanges among developers, the report submitter, and other interest parties. These comments provide valuable insights and additional information related to the reported issue.

2.2 LLM-based bug report representation

After extracting bug reports, we obtain a corpus of text data. To represent these texts effectively, we utilize a powerful pre-trained large language model called *text-embedding-ada-002*. By applying *text-embedding-ada-002* to the texts, we obtain dense and low-dimensional embedding vectors that serve as compact representations of the original bug reports.

Specifically, *text-embedding-ada-002* model employs the Transformer architecture (Ashish et al., 2017) to convert input into a 1,536-dimensional vector. Firstly, each input bug report is tokenized and segmented into tokens. Next, the tokens pass through 96 decoder layers, each comprising a masked multi-head self-attention mechanism and a feed-forward neural network. The multi-head self-attention layer computes self-attention on the input sequential data, generating feature representations for each position in the sequence. The feed-forward network performs fully connected calculations on the feature vectors at each position, producing new feature representations. Its crucial role is to provide nonlinear transformations.

The decoder layers start by applying h different linear projections to the Query, Key, and Value. The resulting attention values for each head i are calculated as follows:

$$head_i = attention(QW_i^Q, KW_i^K, VW_i^V) \tag{1}$$

where Q, K, and V represent the query vector, key vector, and value vector, respectively.

The attention mechanism used in the transformer employs scaled dot-product attention, which can be defined as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{(d_k)}})V \tag{2}$$

where, d_k represents the dimension of the query/key vectors.

The resulting attention values from all the heads are concatenated together, resulting in a single multi-head attention output:

$$MultiHead(Q, K, V) = concat(head_1, ..., head_h)W^O \tag{3}$$

where W^O is a weight matrix used to combine the multi-head attention outputs.

Additionally, the decoder includes an additional masked multi-head self-attention layer. This layer prevents the model from seeing future information during sequence prediction. Hence, the final output of the decoder can be represented as:

$$DecoderLayer(y) = LN(y + M_{MHA(y)} + MHA(y, x) + FFN(y)) \tag{4}$$

where y represents the input sequential data, x refers to the output sequence from the encoder, MHA denotes the multi-head self-attention layer, FFN represents the feed forward layer, LN represents the layer normalization layer, and M_{MHA} signifies the masked multi-head self-attention layer.

Finally, the output of the attention layer undergoes processing through a feed-forward neural network. The position-wise feed-forward network is a fully connected feed-forward neural network where each word at a position passes through the same network independently. It essentially consists of two fully connected layers. After passing through all the decoder layers, the final output is generated by the last decoder layer. This output contains the contextual information of the bug reports and serves as the ultimate embedding vector representation for bug reports. This embedding vector will be used for subsequent classification tasks.

2.3 Bug report classification

In this section, we conduct the bug report classification task at three levels, as depicted in Fig. 2. At the first level, we classify bug reports into two categories: bugs and non-bugs. As depicted in Herzig et al. (2013), not all bug reports contain actual bugs. Therefore, bug reports related to requests for new features or enhancements, documentation issues (e.g., missing information, outdated documentation, or harmless warning outputs), compile-time issues (e.g., cmake errors or linking errors), operator errors or duplicate reports are considered non-bugs and should be filtered out. Based on the complexity of fault activation and/or error propagation conditions, we predict bugs into Bohrbugs (BOHs) and Mandelbugs (MANs) in the second level (Grottke & Trivedi, 2005). Finally, within the MAN category, we further differentiate between aging-related

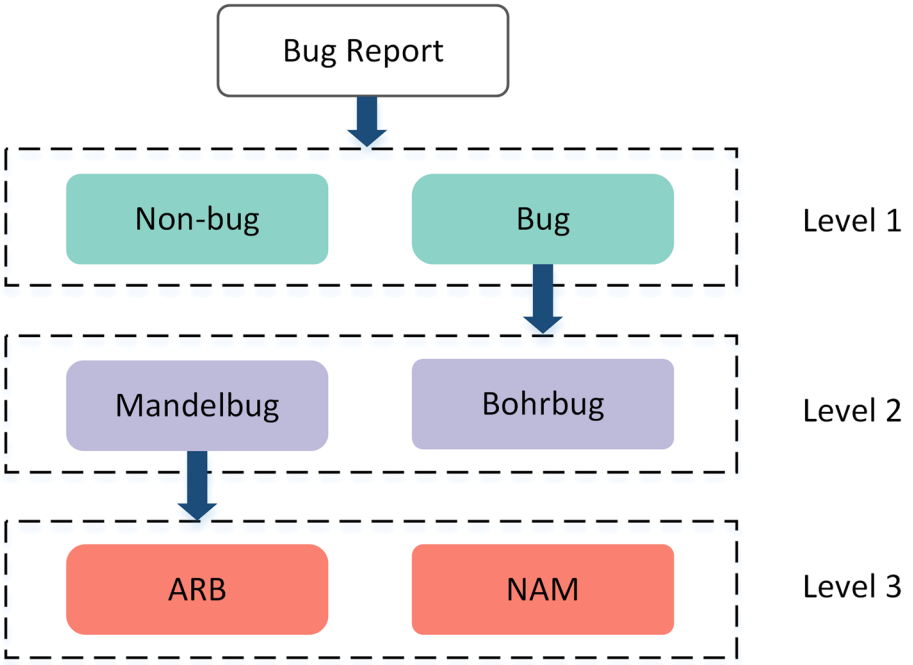

Fig. 2 Classification tasks performed

bugs (ARBs) and non-aging related Mandelbugs (NAMs) depending on whether the bug contributes to the software aging phenomenon. The definitions of BOH, MAN, and ARB are as follows (Du et al., 2022; Cotroneo et al., 2013).

Bohrbug (BOH) A bug that consistently manifests under well-defined conditions. The activation and/or error propagation of BOHs is simple.

Mandelbug (MAN) A bug that cannot always be manifested even under exact conditions. In contrast to BOHs, the activation and/or error propagation of MANs is complex.

Aging-related bug (ARB) A fault that leads to the accumulation of errors either inside the running application or its system-context environment, resulting in an increased failure rate and/or degraded performance.

Non-aging related bug (NAM) If a bug belongs to the MAN category but is not an ARB, it is classified as a NAM.

To perform the classification task, we construct a FFN classifier. FFN is particularly useful for classification tasks because they can learn complex non-linear decision boundaries between classes. By adjusting the weights and biases during training, the network can effectively map input data to the correct output class labels. As depicted in Fig. 1, the network architecture consists of four layers, an input layer, a hidden layer

Table 1 Details of Bug Reports

Project	Level 1		Level 2		Level 3	
	Bug	Non-bug	BOH	MAN	ARB	NAM
TensorFlow	953	1,025	745	157	88	69
MXNET	391	386	270	114	34	80
PaddlePaddle	208	147	152	47	19	28

with 256 neurons, a second hidden layer with 128 neurons, and an output layer with a number of neurons equal to the number of unique classes in the target variable.

The first layer's input shape is determined by the number of features in the input data. The activation function used in both the hidden layers is the Rectified Linear Unit (ReLU), which is commonly used in deep learning models due to its ability to handle non-linear data. As for the output layer, it uses the *Softmax* activation function, converting the layer's output into a probability distribution across different classes.

3 Experimental setup

3.1 Dataset

Table 1 presents the dataset used to evaluate the LLM-BRC framework. This dataset consists of bug reports collected from three popular deep learning frameworks: TensorFlow, MXNET, and PaddlePaddle. Specifically, we collected a total of 1,978 bug reports from the TensorFlow's GitHub repository[1], 777 bug reports from MXNET's GitHub repository[2], and 355 bug reports from PaddlePaddle's GitHub repository[3]. For each bug report, we extracted the title, description and comment components, resulting in a total of 3,110 bug reports. The detailed information of the selected bug reports is presented in Table 1.

3.2 Comparison methods

In this section, we introduce the method used for comparison with our proposed approach, DeepSIM, which is a semantic information-based bug report classification method (Du et al., 2021). DeepSIM is currently the state-of-the-art method for automatically classifying bugs based on fault triggering conditions. In addition, the authors have conducted a comprehensive comparison between DeepSIM and other existing machine learning methods in Du et al. (2021), demonstrating the superior effectiveness of DeepSIM. Consequently, this paper selects DeepSIM as the benchmark for comparison.

DeepSIM: This method begins by training a word2vec semantic model on a vast dataset comprising over two million bug reports collected from 9 open-source software projects. Each word in the text is then transformed into a word vector using the word2vec model. Subsequently, all word vectors are concatenated to form a two-dimension vector representation of the text data. Finally, a convolutional neural network classifier is trained on these text embeddings to predict the categories of bug reports.

[1] https://github.com/tensorflow/tensorflow/issues
[2] https://github.com/apache/incubator-mxnet/issues
[3] https://github.com/PaddlePaddle/Paddle/issues

3.3 Evaluation metrics

In this study, we employ accuracy, precision, recall, and F-measure as evaluation metrics to assess the performance of LLM-BRC. These metrics are calculated based on the confusion matrix, which records the correctly and incorrectly predicted instances for each class. The key components in the confusion matrix are denoted as follows: *TP* for true positive, *FP* for false positive, *FN* for false negative, and *TN* for true negative. The calculation of the selected metrics is as follows.

Accuracy Accuracy is the proportion of correctly predicted instances relative to all instances and is defined as:

$$Accuracy = \frac{TN + TP}{TP + TN + FP + FN}.$$

Precision Precision estimates the proportion of instances correctly predicted as a particular class among all instances classified into that class. This metric assesses the predictive power of the algorithm and is defined as:

$$Precision = \frac{TP}{TP + FP}.$$

Recall Recall measures the proportion of correctly predicted members over all actual class members. It is calculated as:

$$Recall = \frac{TP}{TP + FN}.$$

F-measure The F-measure is a composite metric that combines both precision and recall. It provides a balanced assessment of the algorithm's performance by considering the trade-off between precision and recall. The F-measure is defined as:

$$F - measure = \frac{TP}{TP + 0.5 * (FP + FN)}.$$

We utilize the *K*-fold function from the sklearn library (Pedregosa et al., 2011) to perform cross-validation and randomly split the bug reports into training and test sets. The *K*-fold function divides the dataset into *K* equal-sized folds. During each iteration of the cross-validation process, one fold is used as the test set, while the remaining *K*-1 folds are combined to form the training set. This process is repeated *K* times, and in our experiments, we set $K = 5$.

4 Evaluation and analysis

4.1 Main results

This section presents the experimental results obtained using the LLM–BRC framework for bug report classification in the TensorFlow, MXNET, and PaddlePaddle frameworks.

As shown in Table 2, LLM–BRC LLM—achieves an impressive accuracy of over 92% across all classification tasks within all three deep learning (DL) frameworks. This significant achievement highlights LLM-BRC's ability to effectively extract and learn

Table 2 Classification Results of LLM–BRC

Project	Task	accuracy	precision	recall	F-measure
TensorFlow	bug/non-bug	92.56%	92.46%	93.02%	92.73%
	MAN/BOH	98.47%	98.70%	99.47%	99.08%
	ARB/NAM	98.75%	98.75%	98.75%	98.75%
MXNET	bug/non-bug	94.74%	94.86%	94.94%	94.88%
	MAN/BOH	94.81%	95.31%	96.00%	95.65%
	ARB/NAM	93.91%	95.29%	96.25%	95.76%
PaddlePaddle	bug/non-bug	93.42%	88.71%	89.52%	89.05%
	MAN/BOH	94.36%	96.45%	96.00%	96.20%
	ARB/NAM	92.00%	92.50%	96.67%	94.29%

from features in bug reports. Among all three DL frameworks, the classification results for the TensorFlow framework are the most impressive. For the bug/non-bug classification, LLM–BRC achieves an accuracy of 92.56%. The precision value of 92.46% indicates a high proportion of correctly classified bug reports among all classified bug reports. The recall value of 93.02% suggests that LLM–BRC effectively captures a large portion of actual bug instances. These results contribute to an F-measure of 92.73%, indicating a good balance between precision and recall. In comparison, the PaddlePaddle framework exhibits the most unfavorable classification results. This difference can be attributed to the wider popularity and use of TensorFlow, which has fostered a relatively mature development community. As a result, TensorFlow benefits from a larger number and higher quality of bug reports. PaddlePaddle, on the other hand, is characterized by a smaller corpus of bug reports that tend to be of lower quality. This difference in the dataset directly affects classification results.

> Based on the above results, it is evident that the quantity and quality of bug reports are significant determinants of classification performance. TensorFlow, benefiting from the highest quantity and quality of bug reports, achieves the most precise classification results. In contrast, PaddlePaddle, with fewer and lower-quality bug reports, shows comparatively less accurate classification outcomes.

To further evaluate the effectiveness of the LLM-BRC framework, we conduct a detailed comparative analysis with the state-of-the-art DeepSIM method. Utilizing identical bug reports as input for both LLM-BRC and DeepSIM, the classification results for the TensorFlow dataset are illustrated in Fig. 3. From the results, it is evident that the LLM-BRC framework outperforms the DeepSIM method on all four metrics. Specifically, LLM-BRC exhibits a substantial increase in classification accuracy of 17.21% to 69.15% across three distinct classification tasks when compared to DeepSIM. Notably, in the bug/non-bug classification task, LLM-BRC demonstrates a marked improvement, enhancing accuracy by 69.15%, precision by 68.97%, recall by 68.15%, and F-measure by 69.12%. Overall, these results distinctly highlight LLM-BRC 's consistent effectiveness across various classification tasks, underscoring its potential for practical implementation in the field. The primary factor contributing to LLM-BRC's superior performance is its core integration of an advanced large language model, which underscores the model's adeptness at understanding complex linguistic nuances and producing embeddings that richly encapsulate textual information, thereby driving its superior classification performance.

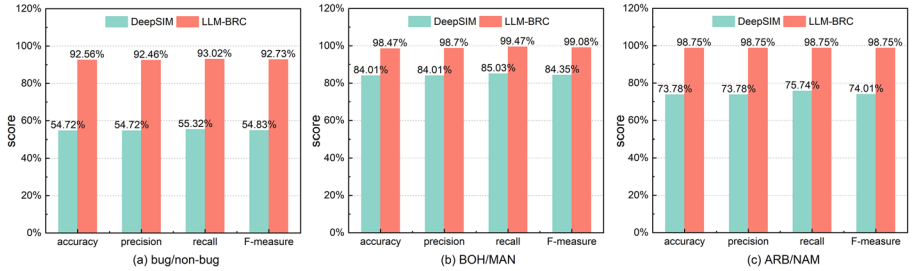

Fig. 3 Comparison of Classification Results between LLM-BRC and DeepSIM

4.2 Ablation study

We conduct ablation studies to analyze the effectiveness of each component in LLM–BRC, including the influence of different components of bug reports, the influence of classifiers and embedding models embedded in LLM–BRC.

4.2.1 Ablation on Bug Report Components

In this section, we investigate the impact of different bug report components on bug report classification results. Bug reports typically consist of various components, including the title, description, and comments. In order to assessing the impact of individual components on the performance of bug report classification, we conducted a series of experiments to isolate and evaluate the contribution of each component (i.e., title, description, and comments) towards overall accuracy.

Analyzing how these components influence bug report classification performance can provide valuable insights into the importance of each component in predicting bugs accurately. We conduct experiments using four different input configurations: ❶title only, ❷description only, ❸comments only, and ❹a combination of the title, description, and comments. In this section, to ensure consistency and enable a fair comparison of the results, all experiments were conducted based on the SVC classifier.

As depicted in Fig. 4, in the context of TensorFlow bug prediction, incorporating information from all bug report components consistently leads to the highest performance across the bug/non-bug, BOH/MAN, and ARB/NAM classification tasks. For the bug/non-bug prediction task, we observe varying levels of accuracy when using different bug report components individually. The accuracy achieved using only the title information is 55.59%,

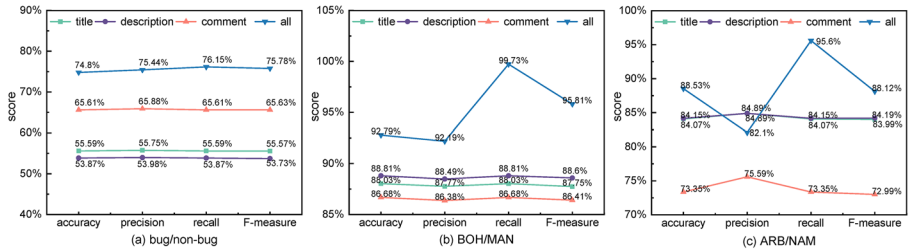

Fig. 4 Classification results based on different bug report components (TensorFlow)

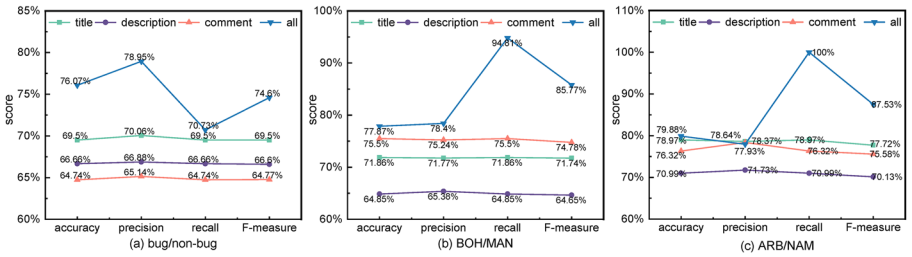

Fig. 5 Classification results based on different bug report components (MXNET)

while leveraging only the description information results in an accuracy of 53.87%. However, incorporating solely the comments demonstrates a significant improvement, reaching an accuracy of 65.61%. Notably, combining all three components as inputs substantially boosts the accuracy to 74.8%. Moreover, employing all bug report components leads to the highest precision (i.e., 75.44%), recall (i.e., 76.15%), and F-measure (i.e., 75.78%) scores. Similarly, for the BOH/MAN prediction task, utilizing all bug report components leads to the best predictive results. The accuracy, precision, recall, and F-measure values are 92.79%, 92.19%, 99.73%, and 95.81%, respectively. Regarding the ARB/NAM prediction task, the utilization of all bug report components results in the highest accuracy (i.e., 88.53%), recall (i.e., 95.6%), and F-measure (i.e., 88.12%) scores. However, the recall values are higher when only using the title and description information.

As for the prediction of MXNET bugs, as shown in Fig. 5, utilizing all bug report components results in the best bug/non-bug and BOH/MAN prediction outcomes. For the bug/non-bug prediction task, the accuracy, precision, recall, and F-measure obtained are 76.07%, 78.95%, 70.73%, and 74.6%, respectively. For the BOH/MAN task, the accuracy, precision, recall, and F-measure are 77.87%, 78.4%, 94.31%, and 85.77%, respectively. Regarding the ARB/NAM prediction, utilizing all bug report components yields the highest accuracy, recall, and F-measure, while the precision obtained is slightly lower than that of using only title and description information.

Regarding the prediction of PaddlePaddle bugs, as shown in Fig. 6, we observed that when performing the bug/non-bug task, the highest accuracy and precision were achieved when utilizing all bug report components. However, incorporating comments leads to a decrease in recall and F-measure. Similarly, for the BOH/MAN prediction, the best predictive resultswere also obtained by utilizing all bug report components, followed by using only comments. As for the ARB/NAM prediction, utilizing all bug report components leads to the highest recall and F-measure scores.

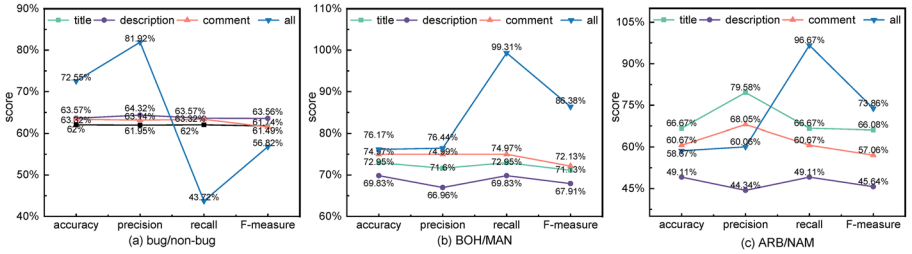

Fig. 6 Classification results based on different bug report components (PaddlePaddle)

In summary, our analysis clearly indicates that each component of a bug report contributes valuable information to the bug report classification process. In addition, it is of utmost importance to consider all available bug report components when applying the LLM-BRC method for bug report classification. By integrating multiple components of information in bug reports, we can effectively enhance the performance of bug report classification models, leading to more reliable and accurate classification results.

4.2.2 Ablation on classifiers

In this section, we compare the FFN classifier embedded in LLM–BRC with 6 traditional machine learning classifiers commonly used in previous studies (Frattini et al., 2016; Du et al., 2017).

As illustrated in Tables 3, 4 and 5, the FFN classifier has the most obvious advantage among the seven classifiers evaluated. Especially for the classification of the PaddlePaddle framework's bug reports, the FFN classifier has the most obvious advantages. Compared with other 6 classifiers, the accuracy of the FFN classifier is 28.25%-48.97%, 23.88%-41.17% and 29.38% higher for bug/non-bug, BOH/MAN,

Table 3 Prediction Results of Different Classifiers (TensorFlow)

Task	Classifier	accuracy	precision	recall	F-measure
Bug/ non-bug	SVC	74.80%	75.44%	76.15%	75.78%
	RFC	67.64%	67.03%	73.41%	70.04%
	LOG	73.38%	73.53%	75.46%	74.48%
	GNB	68.80%	69.49%	70.67%	70.02%
	DTC	57.98%	59.64%	57.58%	58.52%
	KNN	63.01%	65.64%	59.14%	62.19%
	MLP	**92.56%**	**92.46%**	**93.02%**	**92.73%**
Impro		**23.74%-59.64%**	**22.56%-55.03%**	**22.15%-61.55%**	**22.37%-58.46%**
BOH/ MAN	SVC	92.79%	92.19%	99.73%	95.81%
	RFC	86.42%	86.00%	99.87%	92.42%
	LOG	89.82%	89.14%	99.87%	94.20%
	GNB	90.69%	95.07%	93.65%	94.32%
	DTC	80.72%	88.92%	87.70%	88.27%
	KNN	90.91%	92.45%	96.96%	94.64%
	MLP	**98.47%**	**98.70%**	**99.47%**	**99.08%**
Impro		**6.12%-21.99%**	**7.06%-11.00%**	**-0.26%-13.42%**	**3.41%-12.25%**
ARB/ NAM	SVC	88.53%	82.10%	95.60%	88.12%
	RFC	84.70%	83.20%	82.64%	82.81%
	LOG	86.61%	84.44%	85.38%	84.71%
	GNB	83.43%	77.48%	89.89%	82.80%
	DTC	72.58%	70.60%	70.99%	70.02%
	KNN	74.56%	91.28%	47.80%	61.37%
	MLP	**98.75%**	**98.75%**	**98.75%**	**98.75%**
Impro		**11.54%-36.06%**	**20.28%-39.87%**	**3.29%-39.10%**	**12.06%-41.03%**

Table 4 Prediction Results of Different Classifiers (MXNET)

Task	Classifier	accuracy	precision	recall	F-measure
Bug/ non-bug	SVC	76.07%	78.95%	70.73%	74.60%
	RFC	67.64%	67.03%	73.41%	70.04%
	LOG	73.38%	73.53%	75.46%	74.48%
	GNB	73.62%	76.58%	67.62%	71.79%
	DTC	60.88%	60.68%	60.39%	60.40%
	KNN	72.72%	79.52%	61.13%	68.94%
	MLP	**94.74%**	**94.86%**	**94.94%**	**94.88%**
Impro		**24.54%-55.62%**	**20.15%-56.33%**	**34.23%-57.21%**	**27.18%-57.09%**
BOH/ MAN	SVC	77.87%	78.40%	94.81%	85.77%
	RFC	86.42%	86.00%	99.87%	92.42%
	LOG	89.82%	89.14%	99.87%	94.20%
	GNB	78.39%	85.51%	83.70%	84.49%
	DTC	74.49%	79.54%	85.93%	82.56%
	KNN	72.64%	78.80%	83.70%	81.15%
	MLP	**94.81%**	**95.31%**	**96.00%**	**95.65%**
Impro		**5.56%-30.52%**	**6.92%-20.95%**	**-3.88%-14.70%**	**1.54%-17.87%**
ARB/ NAM	SVC	79.88%	77.93%	100.00%	87.53%
	RFC	84.70%	83.20%	82.64%	82.81%
	LOG	86.61%	84.44%	85.38%	84.71%
	GNB	81.58%	82.03%	96.25%	88.17%
	DTC	57.91%	70.66%	68.75%	69.49%
	KNN	72.02%	82.94%	76.25%	78.96%
	MLP	**93.91%**	**95.29%**	**96.25%**	**95.76%**
Impro		**8.43%-62.17%**	**12.85%-34.86%**	**12.73%-40.00%**	**13.04%-37.80%**

and ARB/NAM classification tasks, respectively. Similar trends are observed in the MXNET and TensorFlow frameworks. These results not only verify the effectiveness of the FFN classifier within the LLM–BRC framework, but also indicate its pronounced superiority in handling datasets with smaller sample sizes.

The results clearly show that the FFN outperforms the other 6 classifiers, demonstrating its effectiveness in predicting bugs across different frameworks. These findings underline the potential of deep learning approaches like FFN in bug prediction tasks and their capability to provide substantial performance improvements over traditional machine learning methods.

4.2.3 Ablation on embedding models

In this section, we conduct a comparative analysis of various embedding models, including the OpenAI embedding model *text-embedding-ada-002*, which is employed in our LLM–BRC framework, alongside two other prevalent embedding models: word2vec and BERT. Table 5 provides a detailed overview of these models. Word2vec, notably, has

Table 5 Details of Pre-trained Language Models

Model	Detail	Dimension
word2vec	GoogleNews-vectors-negative300	300
BERT	bert-base-uncased	768
OpenAI	text-embedding-ada-002	1,536

been instrumental in the evolution of natural language processing (NLP) and word embedding technology, introducing distributed word representations that are learned from extensive text data. It has thus become a foundational element in NLP research and applications. For our experiments, we selected the *GoogleNews-vectors-negative300* model of word2vec, generating 300-dimensional vectors, renowned for its simplicity and effectiveness. The BERT model represents a paradigm shift in NLP, utilizing bidirectional training and Transformer architecture to achieve deeply contextualized word embeddings. We utilized the *bert-base-uncased* variant, producing 768-dimensional word vectors. BERT's flexibility for fine-tuning has yielded substantial enhancements across diverse NLP tasks. Lastly, the OpenAI *text-embedding-ada-002* model, utilized in our experiments, generates embeddings of dimension 1,536.

To ensure the consistent experimental conditions, we employ the same SVC classifier and utilize identical bug information, which includes the title, description, and comments of bug reports, throughout this section. By maintaining a consistent setup, we can confidently compare the performance of different language models. In Fig. 7, we present the classification results of the TensorFlow framework, where it is evident that the *text-embedding-ada-002* model consistently outperforms the BERT and word2vec models on all three classification tasks, i.e., bug/non-bug, BOH/MAN, and ARB/NAM. The text-embedding-ada-002 model, compared with the BERT model, improved the classification accuracy by 10.99% to 44.30%, and compared with word2vec, the classification accuracy improved by 9.52% to 36.07% in all three classification tasks. Among the three classification tasks, the *text-embedding-ada-002* model exhibits the most significant advantage for the ARB/NAM classification task. The classification based on the *text-embedding-ada-002* model surpass the BERT model by 44.30% in terms of accuracy and exceed the word2vec model by 36.07%.

> The text-embedding-ada-002 model's evident advantage in the ARB/NAM classification task highlights its exceptional capability to handle datasets with limited samples. This is crucial as obtaining large labeled datasets for bug classification can be challenging and time-consuming. In addition, the text-embedding-ada-002 model is particularly outstanding in terms of recall and F-measure improvements, which

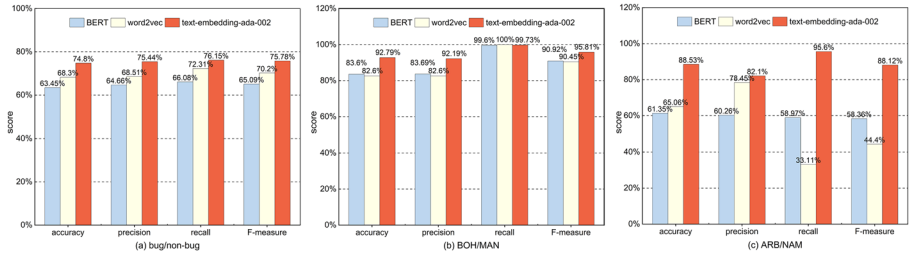

Fig. 7 Comparison of prediction results based on different LLMs

indicates that it excels in effectively addressing the class imbalance issue. These strengths make the text-embedding-ada-002 model particularly beneficial for bug classification tasks, especially when dealing with small datasets and imbalanced class distributions.

5 Threats to Validity

Threats to Internal Validity The quality of bug reports used in this study can affect the internal validity. Since bug reports are written by various users and developers, there may be variations in their quality. Low-quality bug reports, containing incomplete or misleading information, might impact the performance of the proposed method. To mitigate this threat, we adopted the *K*-fold cross-validation technique, which helps reduce the impact of randomness and provides a more robust evaluation of the framework.

Threats to External Validity There is a risk to the external validity of the research, as the classification results obtained from TensorFlow, MXNET, and PaddlePaddle might differ when applied to other frameworks with distinct features. We have appropriately acknowledged this threat and are cautious not to overgeneralize conclusions to all deep learning frameworks, thereby mitigating the potential impact on external validity.

Threats to Construct Validity A threat to construct validity lies in the interpretability of deep learning classifiers. The black-box nature of deep learning models makes explaining their decision-making process challenging. This issue may undermine the credibility of the classification results of our proposed framework. In recent years, researchers have introduced various methods to explain the deep learning process, which may partially alleviate this problem.

6 Related Work

Empirical Study of Bugs in Deep Learning Frameworks In recent years, researchers have made significant progress in analyzing bugs within deep learning frameworks, exploring both general and specific bug types. Typically, the classification criteria for general bugs depend on their underlying causes and impacts. Chen et al. (2022) conducted a study on 1,000 bugs in TensorFlow, PyTorch, MXNet, and DL4J, categorizing them based on 13 root causes and 6 symptoms. Jia et al. (2021) conducted a study on bugs in TensorFlow. They analyzed 202 bug fix submissions, out of which 84 had corresponding bug reports. The authors classified bugs into 11 types based on root causes and identified 6 different bug symptoms. Apart from TensorFlow, Islam et al. (2019) also investigated Caffe, Keras, Theano, and Torch, analyzing 2,716 posts from Stack Overflow and 500 bug fix submissions from GitHub. They identified 6 main root causes and 6 bug symptoms, with crashes being the most common bug symptom in deep learning frameworks. Yang et al. (2022) analyzed 1,127 bug reports from 8 deep learning frameworks, developing a bug classification system to explore root causes and manifestations of bugs from the source code perspective.

Makkouk et al. (2022) investigated performance bugs in TensorFlow and PyTorch, analyzing the proportion difference and underlying root causes of performance and non-performance bugs. Tambon et al. (2021) classified silent bugs in deep learning frameworks. Silent bugs refer to bugs that occur in the framework without causing crashes, hangs, or warnings but still negatively impact the quality of the framework. They collected 77 reproducible silent bugs from TensorFlow's Keras repository and analyzed their occurrences and impact on deep learning programs. Ren et al. (2020) classified and compared system-related bugs in both deep learning frameworks and traditional software. According to their findings, configuration bugs were prevalent in traditional software systems but rare in intelligent computing frameworks. Liu et al. (2022) studied aging-related bugs (ARBs) in TensorFlow, MXNet, PaddlePaddle, and MindSpore by manually screening 138 ARBs out of 13,694 bug reports. Du et al. (2022) performed the first comprehensive empirical study on fault triggering conditions in TensorFlow, MXNet, and PaddlePaddle, analyzing 3,555 bug reports their GitHub repositories.

Automatic Classification of Bug Reports In (Antoniol et al., 2008), Antonial et al. employed the TF-IDF bag-of-words model along with machine learning classifiers to distinguish bugs from other kinds of issues. They conducted their experiments on bug reports from Mozilla, Eclipse, and JBoss, achieving classification results ranging from 77 to 82%. Pingclasai et al. (Pingclasai et al., 2013) applied topic modeling to pre-process bug reports from the HTTP Client, Jackrabbit, and Lucene projects. The resulting F-measure scores were in the range of 66%-76%, 65%-77%, and 71%-82%, respectively. In (Otoom et al., 2019), Otoom et al. aimed to develop a classifier capable of categorizing newly received bug reports into corrective (bug fixing) report, and perfective (major maintenance) report. To achieve this, they proposed a feature set based on keyword occurrences.

In addition to classify bug reports into actual bugs and non-bugs, Wen et al. (2016) proposed CoLUA, which classified bug reports into configuration-related or non-configuration-related. CoLUA first extracted words with higher weights in bug report texts using feature selection algorithms such as information gain and chi-square. Then, they used methods like naive Bayes, logistic regression, and decision trees for automatic classification of configuration bugs. In (Frattini et al., 2016), Frattini et al. automated bug report classification using two Naive Bayes and Bayesian Network classifiers. They categorized bugs into workload-dependent and environment-dependent types based on reproducibility. However, the aforementioned approaches were built on the bag-of-words model, neglecting the semantic information in bug reports. To address this limitation, Du et al. (2022) introduced the DeepSIM method, which combined the word2vec semantic model with a deep learning classifier for automatic Mandelbug classification. Additionally, in Li et al. (2021), the authors proposed the ARB-BERT method, which fine-tuned the BERT model to extract more comprehensive and accurate semantic information from bug report texts, completing the representation and automatic classification of aging-related bugs.

7 Conclusions

In this paper, we presented LLM–BRC, a Large-Language Model-based Bug Report Classification framework. In LLM–BRC, we employed the latest large-language model *text-embedding-ada-002* from OpenAI to capture the semantic information of bug reports and represented each bug report as an embedding vector. Subsequently, we constructed a

Feed-Forward Neural Network as the classifier, utilizing the embedding vectors generated by the LLM as input, thereby obtaining the bug report classification results. We evaluated LLM–BRC on a total of 3,110 bug reports collected from TensorFlow, MXNET, and PaddlePaddle. The experimental results demonstrated the effectiveness of our method in bug report classification for deep learning frameworks, achieving an impressive accuracy range of 92% to 98.75%. Moreover, we investigated various factors that influence bug report classification performance, including components of bug reports, classifiers and different language models. The conclusions contributed to the field of bug report classification and offered valuable guidance for researchers and practitioners in improving the bug report classification performance.

Author contributions Xiaoting Du: methodology and experimental design, and original draft writing. Zhihao Liu: experiments conducting. Chenglong Li: data collection and processing. Xiangyue Ma: data analyzing and results presentation. Yingzhuo Li and Xinyu Wang: verification of results and the manuscript polishing.

Funding This work was supported in part by the Fundamental Research Funds for the Central Universities (2023RC06), and in part by the Open Research Fund of Shanghai Key Laboratory of Trustworthy Computing (East China Normal University).

Data Availability Statement The data of this study is openly available in GitHub at https://sites.google.com/view/llmbp/.

Declarations

Competing interests The authors declare no competing interests.

References

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., & Guéhéneuc, Y. G. (2008). Is it a bug or an enhancement? A text-based approach to classify change requests. *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds* (pp. 304-318).

Ashish, V., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems, 33*, 1877–1901.

Cai, T. M., Feng, D., Liu, S., Liu, S., Kikinis, R., & Pujol, S. (2014). Early diagnosis of alzheimer's disease with deep learning. *IEEE 11th international symposium on biomedical imaging*.

Chen, J., Liang, Y., Shen, Q., Jiang, J., & Li, S. (2022). Toward understanding deep learning framework bugs. ACM Transactions on Software Engineering and Methodology.

Chen, C., Seff, A., Kornhauser, A., & Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. *Proceedings of the IEEE international conference on computer vision* (pp. 2722-2730).

Collobert, R., Bengio, S., & Mariéthoz, J. (2002). Torch: a modular machine learning software library. *Technical report, Idiap*.

Cotroneo, D., Grottke, M., Natella, R., Pietrantuono, R., & Trivedi, K. S. (2013). Fault triggers in open-source software: An experience report. *2013 IEEE 24th International symposium on software reliability engineering (ISSRE)* (pp. 178–187). IEEE.

Du, X., Zheng, Z., Xiao, G., & Yin, B. (2017). The automatic classification of fault trigger based bug report. *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 259–265). IEEE.

Du, X., Zheng, Z., Xiao, G., Zhou, Z., & Trivedi, K. S. (2021). Deepsim: Deep semantic information-based automatic mandelbug classification. *IEEE Transactions on Reliability, 71*(4), 1540–1554.

Du, X., Sui, Y., Liu, Z., & Ai, J. (2022). An empirical study of fault triggers in deep learning frameworks. *IEEE Transactions on Dependable and Secure Computing*.

Frattini, F., Pietrantuono, R., & Russo, S. (2016). Reproducibility of Software Bugs: Basic Concepts and Automatic Classification. *Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi on his 70th Birthday*, 551-565.

Frieder, S., Pinchetti, L., Griffiths, R. R., Salvatori, T., Lukasiewicz, T., Petersen, P., Chevalier, A., & Berner, J. (2023). Mathematical capabilities of chatgpt. *arXiv preprint* arXiv:2301.13867.

Girija, SS. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Software available from tensorflow. org, 39*(9).

Grottke, M., & Trivedi, K. S. (2005). Software faults, software aging and software rejuvenation (special survey: New development of software reliability engineering). *The Journal of Reliability Engineering Association of Japan, 27*(7), 425–438.

Guo, Q., Xie, X., Ma, L., Hu, Q., Feng, R., Li, L., Liu, Y., Zhao. J., & Li, X. (2018). An orchestrated empirical study on deep learning frameworks and platforms. *arXiv preprint* arXiv:1811.05187.

Guo, S., Wang, Y., Li, S., & Saeed, N. (2023). Semantic communications with ordered importance using chatgpt. *arXiv preprint* arXiv:2302.07142.

Herzig, K., Just, S., & Zeller, A. (2013). It's not a bug, it's a feature: how misclassification impacts bug prediction. *2013 35th international conference on software engineering (ICSE)* (pp. 392-401). IEEE.

Islam, M. J., Nguyen, G., Pan, R., & Rajan, H. (2019). A comprehensive study on deep learning bug characteristics. *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering* (pp. 510-520).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *Proceedings of the 22nd ACM international conference on Multimedia* (pp. 675-678).

Jia, Li., Zhong, H., Wang, X., Huang, L., & Xuansheng, Lu. (2021). The symptoms, causes, and repairs of bugs inside a deep learning library. *Journal of Systems and Software, 177*, 110935.

Li, M., & Yin, B. B. (2021). Arb-bert: An automatic aging-related bug report classification method based on bert. *2021 8th International Conference on Dependable Systems and Their Applications (DSA)* (pp. 474–483). IEEE.

Liu, Z., Zheng, Y., Du, X., Hu, Z., Ding, W., Miao, Y., & Zheng, Z. (2022). Taxonomy of aging-related bugs in deep learning libraries. *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)* (pp. 423–434). IEEE.

Lux, M., & Bertini, M. (2019). Open source column: Deep learning with keras. *ACM SIGMultimedia Records, 10*(4), 7–7.

Makkouk, T., Kim, D. J., & Chen, T. H. P. (2022). An empirical study on performance bugs in deep learning frameworks. *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 35–46). IEEE.

Nov, O., Singh, N., & Mann, D. (2023). Putting chatgpt's medical advice to the (turing) test. *medRxiv*, 2023–01.

Otoom, A. F., Al-jdaeh, S., & Hammad, M. (2019). Automated classification of software bug reports. *Proceedings of the 9th international conference on information communication and management* (pp. 17-21).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Pingclasai, N., Hata, H., & Matsumoto, K. I. (2013). Classifying bug reports to bugs and other requests using topic modeling. *2013 20Th asia-pacific software engineering conference (APSEC)* (Vol. 2, pp. 13–18). IEEE.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Jeffrey, Wu., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI Blog, 1*(8), 9.

Ray, P. P. (2023). ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*.

Ren, Y., Gay, G., Kästner, C., & Jamshidi, P. (2020). Understanding the nature of system-related issues in machine learning frameworks: An exploratory study. *arXiv preprint* arXiv:2005.06091.

Tambon, F., Nikanjam, A., An, L., Khomh, F., & Antoniol, G. (2021). Silent bugs in deep learning frameworks: an empirical study of keras and tensorflow. *arXiv preprint* arXiv:2112.13314.

Team, T.T.D., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., et al. (2016) Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint* arXiv:1605.02688.

Wen, W., Yu, T., & Hayes, J. H. (2016). Colua: Automatically predicting configuration bug reports and extracting configuration options. *2016 IEEE 27Th international symposium on software reliability engineering (ISSRE)* (pp. 150–161). IEEE.

Xia, X., Lo, D., Wang, X., & Zhou, B. (2014). Automatic defect categorization based on fault triggering conditions. *2014 19th International Conference on Engineering of Complex Computer Systems* (pp. 39–48). IEEE.

Yang, Y., He, T., Xia, Z., & Feng, Y. (2022). A comprehensive empirical study on bug characteristics of deep learning frameworks. *Information and Software Technology, 151*, 107004.

Zhang, Y., Chen, Y., Cheung, S. C., Xiong, Y., & Zhang, L. (2018). An empirical study on TensorFlow program bugs. *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis* (pp. 129-140).

Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Xiaoting Du is a lecturer at School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications. Her current research interest focuses on quality assurance for intelligent software systems and software repository mining.

Zhihao Liu received his bachelor degree and master degree from Beihang University in 2020 and 2023 respectively. He is currently a doctoral student at Beihang University. His research interests include intelligent software reliability and AI4SE.

Chenglong Li is a PhD student at School of Automation Science and Electrical Engineering, Beihang University. His current research interest focuses on software engneering, software reliability and software fault localization.

Xiangyue Ma received his Bachelor's degree from Qingdao University in 2016. He is pursuing his Doctoral degree in the School of Automation Science and Electrical Engineering at Beihang University. His research primarily focuses on intelligent software testing and software testing using large language models.

Yingzhuo Li an undergraduate in the class of 2022, is currently studying Software Engineering at Beijing University of Posts and Telecommunications. She currently has a strong interest in deep learning and big data.

Wang Xinyu is now a second-year undergraduate at School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications in China, majoring in software engineering. She is interested in Artificial Intelligence, especially Natural Language Processing.