A Transformer-CNN based Multi-Class Bug Report Classification with Explainable Evaluation

Jammy Chan
masterdrm@berkeley.edu

Sankalan Bhattacharjee
sankalan@berkeley.edu

***Abstract*—In this work, we propose the task of automatically classifying bug reports, which is critical for any business to ensure a positive customer experience. While such user feedback or reports are generally represented using natural languages, due to frequent occurrences of technical jargons the task of categorizing each report accurately is challenging. In this paper, we explore the feasibility of various state-of-the-art language models in designing an artificially intelligent software prototype system that not only predicts the class membership for each input technical report, but also provides an in-depth explanation to support each of its decisions. First, we design a large language model-based summarization module that can effectively summarize the main report content. Second, we leverage several transformer-based pre-trained embedding modules to derive a feature descriptor of the summarized content, which is fed as an input to the following multi-class convolutional neural network to predict its category details. Third, the LIME-powered explanation module enables the system to justify each of its decisions, which not only helps understand the system reasoning behind a decision but also evaluates the system's class-specific relative categorization ability in parallel. Performances and comparative analyses on a public bugs dataset suggest that both summarization and embedding modules are equally important to ensure reliable system performance, while the consequential explanation module ascertains enhanced decision transparency.**

***Index Terms*—Language Model, transformer, Classification, explainability**

I. Introduction

Software bug classification is an important task in software development and product management industries. This involves categorizing software or product-related issues, described in natural language-based reports, into predefined groups. More specifically, bug reports capture the user requirements and complaints about a product. Therefore, due to the sensitivity of the problem scenario, an efficient team assignment and accurate prioritization scheme are critical to facilitate smooth bug resolutions, which may minimize bug resolution time and operational costs. In fact, timely resolution of these issues is extremely important to ensure an improved user experience and success of the product in the market. However, typically a software ticket describing a bug also involves many technical terms as well as console or log outputs. Those tokens are cryptic compared to traditional vocabulary words. In addition, some of the vocabulary spans across multiple categories or is too generic. Thus, due to the noisy and unstructured data characteristics and the lack of reliable annotations, the task of bug classification is labor-intensive. For example, irrelevant bug reports may be accepted and implemented within the updated system, while useful comments may be ignored, which could have been important to maintain the competitiveness of the product in the market.

To address this task, existing works have leveraged the power of machine learning-based models to design several automation schemes for ticket resolution [1], [2]. While traditional machine learning models have demonstrated reasonable promise, sophisticated deep learning-based architectures have improved the results significantly [3]. A set of recent literature also proposes human-computer-interaction mechanisms to ensure appropriate expert intervention, based on the nature of the bug report. A common practice is to classify tickets into multi-level hierarchy, where each level of the hierarchy describes the issue at different levels of detail [4], [5]. Various predictive models attempt to predict the resolution status of newly submitted bug reports [6], [7].

This work explores the task of bug categorization as a text classification task, with the following objectives:

1) Building Transformer-CNN model, which combines the knowledge of the pre-trained transformer-based embedding model and Convolutional Neural Network (CNN) to enable an accurate classification of the multiple bug categories
2) Leveraging pre-trained model to obtain summarized text as input to the Transformer-CNN Model results in improved precision decision. The summarization as a part of pre-processing of the raw input comment is not only proved more effective but also efficient in parallel.
3) A detailed experimental analysis combined with an initial explainability analysis using Local Interpretable Model-Agnostic Explanation (LIME) to evaluate the model performance

II. Related Works

Due to a wide variety of bug reports describing multiple types of software issues and an increasing value of report generated daily, the task of bug report classification is both tedious and difficult. Thus, while manual categorization is not infeasible, over the last decade, researchers have extensively explored the option of automating the task by leveraging sophisticated machine-learning models [8]. The earliest bug classification method is Orthogonal Defect Classification (ODC), which was designed by IBM's Chillarege et al. [9] in 1992. Researchers have adopted traditional machine learning models like Decision Trees, Naive Bayes and Logistic Regression classifiers to automatically identify the presence of a bug issue in

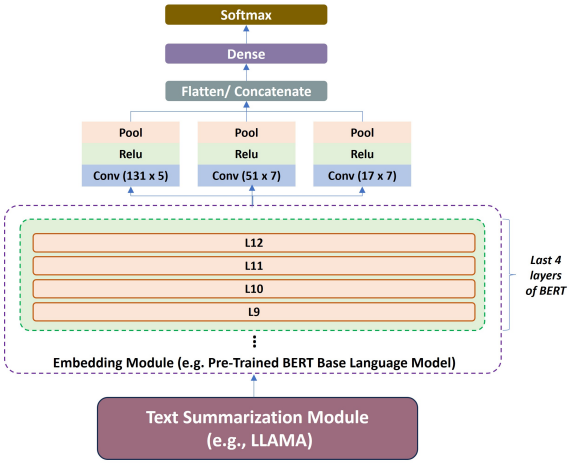

Fig. 1. Model Architecture Overview

a report [10], [11]. Boser et al. [12] use a simple word-count-based Term Frequency-Inverse Document Frequency (TF-IDF) [13] text representation technique to classify tickets using Support Vector Machine (SVM). Pingclasai et al. [14] design a classifier to identify the authenticity of bugs by leveraging the topic model of Latent Dirichlet Allocation (LDA) in combination with Naive Bayes and Regression models.

A set of recent works have also leveraged sophisticated deep neural networks, such as Multi-layer Perceptrons [15], Convolutional Neural Networks [4], and Recurrent Neural Networks [16] to solve the task of report classification. In fact, the task of bug report classification can also be explored in the broader context of text classification which has experienced revolutionary progress with the advent of transformer architecture. Due to its effective noise-robustness, Bidirectional Encoder Representations from Transformers (BERT) [17] have been extensively used for text classification and in particular for bug report classification [2] as well. In this project, we explore various Large Language Models(LLM) for text summarization and several transformer-based embedding techniques to represent the summarized report, an effective report representation scheme that not only ensures noise robustness but also delivers a compact data descriptor, which may effectively facilitate the following categorization objective using Convolutional Neural Network based classifier.

A. Classification Framework

In this work, we design a classification framework that utilizes pre-trained BERT models [17] with Convolutional Neural Networks (CNN) [18] to classify an input report into multiple bug categories.

III. Methodology

In this work, we design a bug classification model for the categorization of a ticket that describes issues related to software bugs, with different bug types. In contrast to the existing baseline model by Marcuzzo et al. [5] that uses the entire comment of a report as input to the multiclass classifier, we leverage a summarized version of the comment to represent each report in the dataset. While this helps to control the size of the natural language-based input to the proposed classifier, important to note that the quality of the summarization holds the key to delivering a successful classification performance on such summarized input. Figure 1 provides an overview of the proposed BERT-CNN classifier. As illustrated in the figure, the summarized comment is passed through a transformer-based embedding layer. The resulting output is then processed by 3 parallel CNN layers each followed by a max-pooling layer and 2 Fully Connected (FC) layers, of which the last layer is the Softmax. In the next section we will report an extensive comparative study to evaluate the model performances by adopting several state-of-the-art Large Language Based summarization techniques.

A. Data Preprocessing via Text Summarization:

As a part of pre-processing, each input report comment is summarized using different language models, which include the followings:

- Text-to-Text Transfer Transformer (T5) [19] is a simple yet effective large-scale transformer-based language model that has attained state-of-the-art performances in various Natural Language Processing (NLP) tasks, including text summarization. With an encoder-decoder model that is pre-trained for multiple NLP tasks (both supervised and unsupervised) by designing a multi-task learning framework. An important advantage of T5 is that it can serve different tasks just by simply changing the prefix of the input. For example, to execute a translation task, the required input will have to be like "translate English to French: ..." whereas for summarization, the input format is required to be slightly different as "summarize: ...".
- The LLaMA (Large Language Model Meta AI) [20] is a state-of-the-art foundational large language model that is pre-trained on a large set of unlabeled data, works by utilizing a sequence of input words to predict the next word in the sequence and recursively generate a text. It enables fast computation and requires comparably less computing power and resources to test new approaches, validate others' work, and explore new use cases. LLaMA is available in several sizes (7B, 13B, 33B, and 65B parameters). In our experiments, we have used LLAMA 7B.
- Pegasus is a large Transformer-based encoder-decoder model, pre-trained on massive text corpora with a new self-supervised objective. In PEGASUS, important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. PEGASUS model is evaluated on 12 downstream summarization tasks spanning news, science, stories, instructions, emails, patents, and legislative bills.

1) Embedding Layer: While there are several language models such as ELMo, word2Vec, and GloVe, which are designed to extract features for the downstream classification task, these models are unable to maintain contextual relationships between the words. To address this issue, transformer-

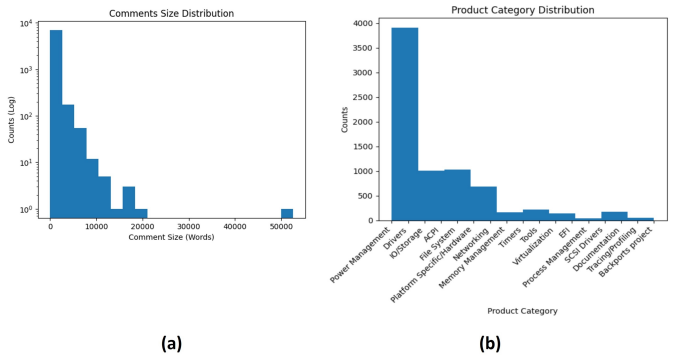

Fig. 2. (a) The graph represents the distribution of the comment sizes and (b) represents the sample distribution for the Bug tickets from different categories.

based encoder-decoder architecture utilizes a self-attention mechanism and feed-forward network to efficiently learn the long-term dependency information. Thus, each input report is converted into a sequence matrix $\mathbf{R} \in \mathbb{R}^{l_r \times d}$, where l_r is the word length of the summarized report and d is the BERT embedding dimension. In this work, we use multiple transformer-based text embedding modules, which include the following:

- The Bidirectional Encoder representation from Transformer (BERT)-base-uncased model [17] to derive the text embedding of an input text.
- XLNET(Transformer-XL) [21]: Unlike Bert which uses Masked Language Model (MLM), XLNet uses Permutation Language Modeling, where the model is trained to predict a sequence of words regardless of their original order. This helps capturing bidirectional context and dependencies similar to bi-directional LSTM. The model uses autoregression factorization to capture sequence dependencies beyond a fixed context window like GPT. In this project, we evaluate xlnet-base-cased model from HuggingFace.
- RoBERTA [22] is an optimized version of Bert model. Features like dynamic masking and the removal of the next sentence prediction (NSP) contribute to better generalization and effectiveness in capturing language patterns because understanding the relations between consecutive sentences is not applicable for the classification task in this project. There are various model favors of Roberta and we choose roberta-base in this project.

2) Convolution Layer: As a primary building block of the classifier, the basic functionality of a CNN is similar to the visual cortex of the animal brain. While CNN models have been predominantly used for analyzing visual data [23]–[26], its application to the text classification task is also very similar and has attained promising results. The only difference is that instead of using pixel values while handling visual data, in the text classification task [18], [27], we use the matrix of word embeddings. CNN layers are specifically designed to extract low-level features while explicitly controlling the length of dependency. This layer computes the convolution of the input matrix with N different randomly generated filters. Assuming a convolution filter $\mathbf{Q} \in \mathbb{R}^{l_h \times d}$, the Convolutional operation is performed repeatedly on $\mathbf{R}$ to obtain output sequence matrix $\mathbf{C} \in \mathbb{R}^{l_r - l_h + 1 \times d}$, where

$$\mathbf{C}(i) = f(\mathbf{w}_c^T \mathbf{R}_{i:i+l_h-1} + \mathbf{b}) \tag{1}$$

The term $\mathbf{R}_{i:i+h-1}$ is the submatrix generated as an output by convolving $\mathbf{R}$ with $\mathbf{Q}$, $\mathbf{b}$ is the bias vector, and f is the non-linear activation function of Convolutional operation. Rectified Linear Unit (ReLU) is used as a non-linear activation function because it can improve the learning dynamics of the network and significantly reduce the number of iterations required for the convergence in deep neural networks.

3) Pooling Layer: The main objective of a pooling layer is to summarize the outputs from the CNN layer to capture semantic details via different feature maps, which are presented via different convolution filters, and pooling layer enables generating a fixed length vector. In this work, we have also utilized the pooling function with max-pooling operator, by which all output feature maps created by the convolution layer are further aggregated to generate a compact final vector representation of the input report.

4) Dense Layers and Classification head: The pooling layer output $\mathbf{x}_r$ is fed into a Neural Network model consisting of two fully connected (FC) layers with rectified linear unit (ReLU) activation function. The activation of the first FC layer is fed into a softmax layer to obtain the probabilistic category membership scores for the incoming signal's anomaly score. While adding more layers makes the network more expressive, it simultaneously becomes harder to train due to increased computational complexity, vanishing gradients, and model over-fitting. The standard back propagation algorithm is employed to update the fully connected layer weight parameters[1]. More specifically, $\mathcal{L}$ denotes the loss function defined as follows:

$$\mathcal{L}(\mathbf{W}) = -\frac{\sum_{j=1}^{|\mathcal{D}|}(\mathbf{1}(y_j = y))log(p(y_j = y|\mathbf{x}_j; \mathbf{W})}{|\mathcal{D}|} \tag{2}$$

where $\mathbf{1}$. is the indicator function, $\mathbf{W}$ represents the neural network weight parameters and $log(p(y_j = y|s_j; \mathbf{W})$ computes the probabilistic score of the sample $\mathbf{x}_j$ for the class $y \in \mathcal{Y}$. The learning task is formulated as solving the minimization problem defined as: $\min_{\mathbf{W}} \mathcal{L}(\mathbf{W})$.

IV. Experiments

A. Dataset Description

Following the experiment protocol described by Marcuzzo et al. [5], in this work, we use the publicly available from Kernel.org Bugzilla to obtain Linux Bugs dataset. It contains tickets annotated with multiple labels, presenting the hierarchical labels of details (e.g., 'product' specifics like Network, Drivers, etc. and "component" specifics like BIOS, scheduler, etc.) regarding the ticket content. The reports are presented using 'comments' (audit trail like, zero or more). The product details of a ticket is used as its only ground truth label. To evaluate the impact of class imbalance in the designed

[1]https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37364585

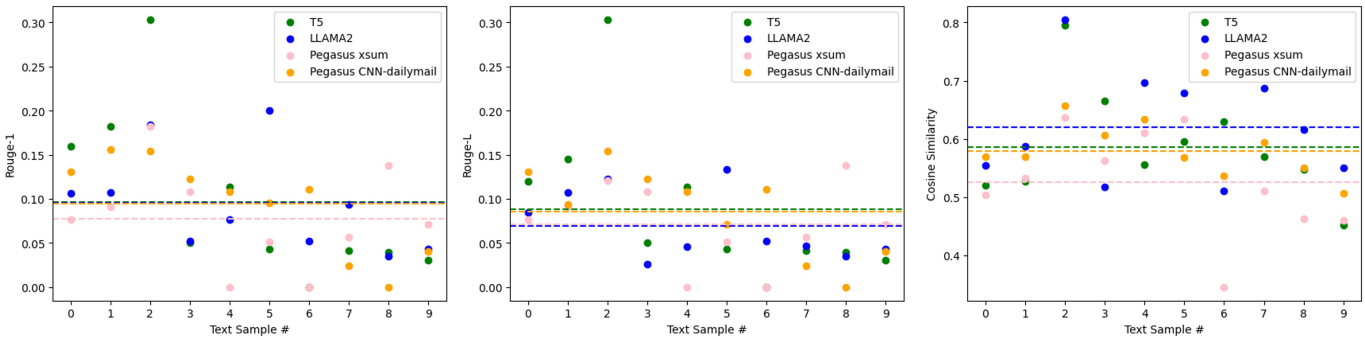

Fig. 3. The ROUGE-1, ROUGE-L, and Cosine similarity scores for the summarized candidate with a ticket topic as the reference. We have chosen 10 random topic samples to generate these results. The dotted lines indicate the average cosine similarity or Rouge scores of each model. In these plots we have reported the performances using four summarization techniques: T5; LLLAMA2; Pegasus xsum; and Pegasus CNN-dailymail.

model, we perform two types of experiments. In the first set of experiments, we include only those classes for which there are at least 100 reports in the dataset. In another experimental setting, we include all classes without taking into consideration of the sizes of class-specific populations. The topic of the report is used as the summarization ground-truth for the input comment. The comment size distribution is heavily skewed as shown in Fig. 2(a).

The incomplete bug reports without any comments or valid labels are discarded. The remaining collection is further refined based on their specific ticket status: 'assigned'; 'reopened'; 'resolved'; 'verified'; 'deferred'; and 'closed'. We also skipped 'new' and 'rejected' status because those tickets might not be classified as a bug after reviews. The resulting collection is comprised of 7419 bug reports written in natural language and annotated with 17 ground-truth labels to be used for the following classification task and the output product class distribution is shown in Fig 2(b). In most software ticketing systems, comments cover bug descriptions, debugging information and progress details as audit trails. Therefore, there will be variety of comment sizes in each bug ticket. Summarization is a useful technique to normalize the text size for the classification model later.

B. Results

The performance evaluation is based on ten randomly selected samples from the training data set. As observed in figure 3, LLAMA2 demonstrates the superior performance on average, while T5 reportedly performing as the second best. The other LLM models (e.g., Pegasus xsum [2], and Pegasus CNN-dailymail [3]) do not commonly show a competitive performance for this dataset. As we delve into the details, this is possibly because of the presence of the cryptic console outputs or logs in the bug comments, which are hard to decode for these models.

Table I represents the comparative performances of a neural network-based model that uses input reports represented using Full comment, Summarized comment, and short topic (synopsis). The model uses a single convolution layer followed by a layer of batch normalization and maxpooling to process the input that is then fed to sequence of two fully connected layers, the first of which is a hidden layer with 512 units and the second is a Softmax layer. As observed in the table, the proposed model demonstrates comparable performances using both forms of comments, i.e. summarized comment and the full comment. Interesting to observe is, although the summarized comment has a significantly reduced embedding size compared to that of the full comment, the proposed model reports an equivalent test accuracy and a slightly better test F1 score. As we observe, the training accuracy and F1 score do not report a noticeable variance, while the validation accuracy and F1 score show improvement compared to that obtained using summarized content to describe a bug report. However in the test environment, the performance of the proposed model deteriorates considerably, when we use only the short topic (i.e. the synopsis of the report) as the input.

Table II represents the comparative performances of the proposed bug classification model using different transformer-based embedding techniques. We have used Accuracy, Loss, and F1 scores to report the results [28]. In this set of experiments, we have chosen different embedding techniques and the text summarization is performed by T5, which was found as a best performing summarization model for our data. In each experiment, the input comment is summarized using T5 and the resulting natural language summarized input is passed through an embedding layer followed by a sequence of 2 fully connected layers, the first of which is a hidden layer with 512 units and the second is a Softmax layer. In the entire set of experiments, we have used a maximum sequence length as 256, the learning rate is 10^{-5}, and the batch size is 8. An early drop epoch is used for each model training based on validation loss (with patience set to 2). As observed in the table, while BERT demonstrates a comparably more reliable performance than XLNet, across all evaluation metrics, Roberta appears to be the most reliant summarization model among all three models we have used in this work. In fact, while Roberta/CNN (i.e. in this experimental setting, Roberta works a embedding

[2]https://github.com/google-research/pegasus
[3]https://metatext.io/models/google-pegasus-cnn_dailymail

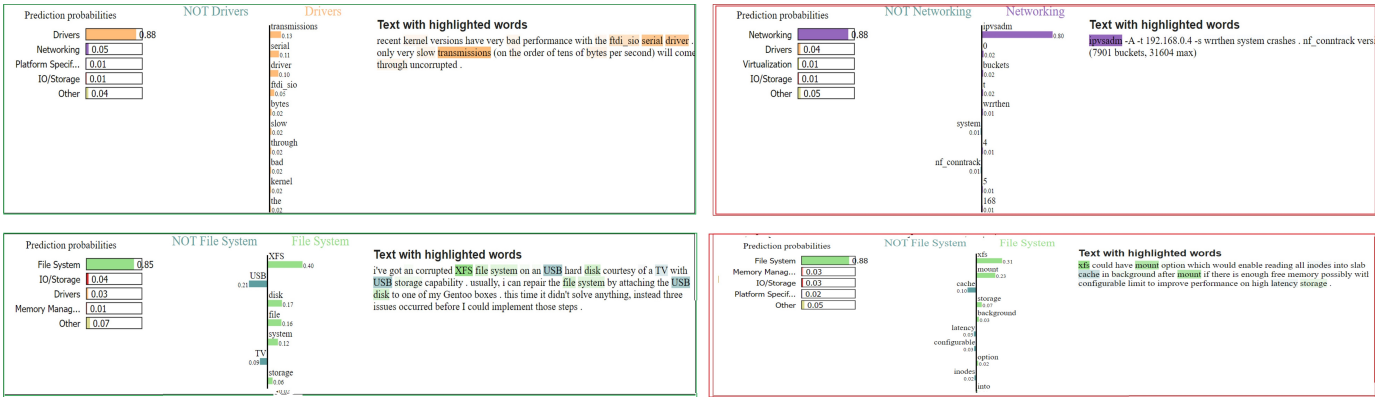

Fig. 4. The examples in the left (and right) column show the explainability evaluations of two queries, which are correctly (and incorrectly) classified by the proposed classification model using Local Interpretable Model-Agnostic Explanations (LIME) [28].

Corpus Type	Embedding Size	Tr. Acc	Tr. F1	Val. Acc	Val. F1	Te. Acc	Te F1
Full comment	1024	**0.9998**	**0.9999**	**0.7287**	**0.4166**	**0.4407**	0.3412
Summarized comment (T5)	256	0.9996	0.9988	0.5914	0.2490	**0.4407**	**0.3458**
Short Topic	256	0.9979	0.9958	0.6807	0.3502	0.3686	0.2900

TABLE I

THE COMPARATIVE PERFORMANCES OF A CNN-BASED MODEL WITH THAT USES INPUT REPORTS REPRESENTED USING FULL COMMENT, SUMMARIZED COMMENT, AND SHORT TOPIC (SYNOPSIS). WE USE WORD EMBEDDING (TENSORFLOW 2.0 IMPLEMENTATION [29]) TO REPRESENT EACH TEXT INPUT INTO A DENSE VECTOR OF FIXED SIZE.

module and the output of which is fed to a CNN based classifier) displays a more robust generalization ability by reporting an improved performance in both validation and test environments, XLNet as an embedding module combined with CNN (denoted as XLNet/CNN) seems to be less reliable compared to both Bert/CNN and Roberta/CNN because the training dataset is based on news articles by journalists at CNN news and the Daily Mail but not the bug reports with software related technical vocabularies.

Table III compares the performance of the proposed model by using different transformer-based embedding modules, where a concatenation of the Summarized comment and short topic (synopsis) as input. As observed in the table, while BERT and Roberta present nearly equivalent performance in the test environment, Roberta seems to be slightly more accurate in terms of predicting the under-represented classes, and thereby reporting a comparably improved F1 score than BERT.

C. Explainability Analysis

The proposed explanation visualization module uses Local Interpretable Model-Agnostic Explanations (LIME)[4] to explain the decision made by the proposed classifier. Using Lime, we found the words that contributed most to the final decision process. Figure 4 shows four example explanation results used to interpret the classification decision of the system. In addition to the category-specific predictions for the entire input report, the explanation module also shows the contribution of each of its words in supporting the decision (e.g., 'Not Drivers' Vs. 'Drivers' or 'Not File system' Vs. 'File System') for the most probable class using a histogram. For example, as shown in the example in the first row of the left column, the summarized report *recent kernel versions have very bad performance with the ftdisio serial driver. only very slow transmissions (on the order of tens of bytes per second) will come through uncorrupted* has found some of the most influencing words, supporting its membership prediction for the 'Drivers' class are: 'serial'; 'driver'; 'transmission'. Similarly in the second row of the left column, words like 'XFS', 'disk', 'file', 'system', and 'storage' contribute positively toward the category 'file system'. However, words like 'USB' an 'TV' contribute negatively for the same category prediction. The mis-classification examples in the right column also describe the reasoning behind the system's prediction. The top row of the right column shows a misclassified sample report, which is an instance from the category 'IO/Storage'. However, due to the presence of the words like 'ipvsadm' the report was misclassified as 'Networking'. Another similar misclassified sample is also highlighted in the second row of the right column. In this example scenario, the sample is misclassified as 'File system', while the ground truth is 'SCSI-Drivers', which is an under-represented category and also semantically similar to the predicted 'File system' category. Furthermore, we note that in both these examples shown in the right column, some technical words, which were not correctly interpreted by the system, have influenced the wrong decision.

[4]https://github.com/marcotcr/lime

Model	Tr. Acc	Tr. Loss	Val Acc	Val Loss	Te. Acc	Te. F1
Bert	0.7277	0.8942	0.6327	1.3316	0.6139	0.5619
Roberta	0.7340	0.8579	0.6318	1.2721	0.6314	0.5975
XLNet	0.6986	0.9665	0.6133	1.3682	0.5903	0.5791
Bert/CNN	**0.8104**	**0.6274**	**0.6310**	1.2814	0.6267	0.5775
Roberta/CNN	0.7843	0.6928	0.6293	**1.2337**	**0.6327**	**0.5983**
XLNet/CNN	0.7317	0.8437	0.5880	1.3850	0.5957	0.5860

TABLE II

THE COMPARATIVE PERFORMANCES OF THE PROPOSED CLASSIFICATION MODEL USING DIFFERENT EMBEDDING TECHNIQUES, WHERE TR. LOSS REPRESENTS THE TRAINING LOSS, VAL LOSS REPRESENTS THE VALIDATION LOSS, AND TE. LOSS REPRESENTS THE TEST LOSS.

Model	Tr. Acc	Tr. Loss	Val. Acc	Val. Loss	Te. Acc	Te F1 Score
BERT/CNN	**0.8966**	**0.3609**	**0.7388**	**0.8736**	0.7264	0.7126
Roberta/CNN	0.8717	0.4095	0.7228	0.9162	**0.7278**	**0.7232**
XLNet/CNN	0.7906	0.6733	0.7102	0.9633	0.6139	0.6378

TABLE III

THE COMPARATIVE PERFORMANCES OF THE PROPOSED MODEL THAT USES A CONCATENATION OF THE SUMMARIZED COMMENT AND SHORT TOPIC (SYNOPSIS) AS INPUT. WE COMPARE THE PERFORMANCE OF THE MODEL USING DIFFERENT EMBEDDING MODULES (E.G., BERT, ROBERTA, XLNET).

V. CONCLUSION

This work explores the problem of automatically classifying bug reports using state-of-the-art large language model (LLM) based embeddings into a convolution layer network. As observed, both pre-trained embedding and the summarization modules are critical components to the proposed categorization framework and play an important role in delivering an accurate classification decision. In fact, compared to the user-provided topic as the report summary, the report summary generated by the LLM models are proved to be more compact and comprehensive in capturing the category-specific information, which facilitates the consequential classification task. The performance further improves if we use both the topic and the summarized report together to represent a report. LIME proves itself as an effective tool to explain each classification decision, which is particularly useful to analyze the misclassified samples. In future, we would like to extend the model to leverage the LIME explanations to automatically flag any ambiguous bug tickets and request users to clarify. Furthermore, to improve the model performance specifically in under-represented category an active learning based Human-Computer-Interactive model would be instrumental, which may utilize sparse human interactions to improve the model's performance iteratively and thus ensure model effectiveness and adaptability in parallel.

REFERENCES

[1] S. Fuchs, C. Drieschner, and H. Wittges, "Improving support ticket systems using machine learning: A literature review," 2022.

[2] S. S. Ali Zaidi, M. M. Fraz, M. Shahzad, and S. Khan, "A multiapproach generalized framework for automated solution suggestion of support tickets," *International Journal of Intelligent Systems*, vol. 37, no. 6, pp. 3654–3681, 2022.

[3] M. A. Arshad and H. Zhiqiu, "Using cnn to predict the resolution status of bug reports," in *Journal of Physics: Conference Series*, vol. 1828, no. 1. IOP Publishing, 2021, p. 012106.

[4] P. Zicari, G. Folino, M. Guarascio, and L. Pontieri, "Discovering accurate deep learning based predictive models for automatic customer support ticket classification," in *Proceedings of the 36th annual ACM symposium on applied computing*, 2021, pp. 1098–1101.

[5] M. Marcuzzo, A. Zangari, M. Schiavinato, L. Giudice, A. Gasparetto, A. Albarelli *et al.*, "A multi-level approach for hierarchical ticket classification," in *Proceedings of the Eighth Workshop on Noisy User-generated Text (W-NUT 2022)*. Wei Xu, Afshin Rahimi, Alan Ritter, Tim Baldwin, 2022, pp. 201–214.

[6] Z. A. Nizamani, H. Liu, D. M. Chen, and Z. Niu, "Automatic approval prediction for software enhancement requests," *Automated Software Engineering*, vol. 25, pp. 347–381, 2018.

[7] Q. Umer, H. Liu, and Y. Sultan, "Sentiment based approval prediction for enhancement reports," *Journal of Systems and Software*, vol. 155, pp. 57–69, 2019.

[8] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *The Computer Journal*, vol. 59, no. 5, pp. 741–773, 2016.

[9] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

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

[11] A. Kukkar and R. Mohana, "A supervised bug report classification with incorporate and textual field knowledge," *Procedia computer science*, vol. 132, pp. 352–361, 2018.

[12] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[13] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[14] N. Pingclasai, H. Hata, and K.-i. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *2013 20Th asia-pacific software engineering conference (APSEC)*, vol. 2. IEEE, 2013, pp. 13–18.

[15] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 406–409.

[16] 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.

[17] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of naacL-HLT*, vol. 1, 2019, p. 2.

[18] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[19] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[20] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[21] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[23] N. Jmour, S. Zayen, and A. Abdelkrim, "Convolutional neural networks for image classification," in *2018 international conference on advanced systems and electric technologies (IC_ASET)*. IEEE, 2018, pp. 397–402.

[24] H. A. Al-Najjar, B. Kalantar, B. Pradhan, V. Saeidi, A. A. Halin, N. Ueda, and S. Mansor, "Land cover classification from fused dsm and uav images using convolutional neural networks," *Remote Sensing*, vol. 11, no. 12, p. 1461, 2019.

[25] I. Strumberger, E. Tuba, N. Bacanin, M. Zivkovic, M. Beko, and M. Tuba, "Designing convolutional neural network architecture by the firefly algorithm," in *2019 International Young Engineers Forum (YEFECE)*. IEEE, 2019, pp. 59–65.

[26] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, "Medical image classification with convolutional neural network," in *2014 13th international conference on control automation robotics & vision (ICARCV)*. IEEE, 2014, pp. 844–848.

[27] M. Z. Amin and N. Nadeem, "Convolutional neural network: text classification model for open domain question answering system," *arXiv preprint arXiv:1809.02479*, 2018.

[28] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.
[29] TensorFlow, "Tensorflow embeddings." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding