Enhancing Software Maintenance with Automated Issue Classification: A Machine Learning Approach

Bhavay Malhotra*
School of EECS
University of Ottawa
Ottawa, Canada
bmalh018@uottawa.ca

Tanishk Nandal*
School of EECS
University of Ottawa
Ottawa, Canada
tnand033@uottawa.ca

Ajay Samuel Victor*
School of EECS
University of Ottawa
Ottawa, Canada
avictor@uottawa.ca

***Abstract*—The exponential growth of open-source software projects has led to an increasing volume of issues reported on platforms like GitHub. These issues span diverse categories, such as bug reports, feature requests, and general inquiries, posing challenges for efficient triaging and classification. Manual classification is labor-intensive and error-prone, especially for large-scale repositories. This study explores automated issue classification using machine learning (ML) and deep learning (DL) techniques, leveraging contextual language models such as RoBERTa and Sentence Transformers for feature extraction. Key contributions include the development of a framework integrating advanced embeddings with classifiers like Support Vector Machines (SVM), Random Forests (RF), LightGBM, XGBoost, Long Short-Term Memory networks (LSTM), and DistilBERT. Empirical evaluations on a balanced dataset of 3,000 GitHub issues demonstrate that DistilBERT, paired with RoBERTa and Sentence Transformer embeddings, achieves near-perfect classification performance (F1-score: 0.992). This work highlights the trade-offs between computational efficiency and accuracy, and underscore the transformative potential of pre-trained transformer models for issue classification. The study establishes a foundation for future research, advocating for enhanced dataset diversity and the exploration of lightweight, domain-specific transformer models. The code implementation for this study can be found here.**

***Index Terms*—Issue classification, Machine learning, Sentence-Transformers, DistilBERT**

I. Introduction

The rapid growth and adoption of open-source software have led to an exponential increase in the number of issues reported on platforms like GitHub. These issues encompass a variety of categories, including bug reports, feature requests, and general inquiries. Efficiently triaging and classifying these issues is critical for effective software project management, as it enables developers to prioritize tasks and allocate resources effectively.

Traditionally, manual classification of GitHub issues has been a time-intensive and error-prone process, particularly when dealing with large repositories that generate hundreds of new issues daily. To address this challenge, researchers have explored the application of machine learning (ML) and deep learning (DL) techniques to automate issue classification. These approaches leverage advanced models, such as transformer-based architectures like BERT [2] and RoBERTa [8], to analyze and categorize issue descriptions with remarkable accuracy.

This paper investigates the performance of various machine learning and deep learning techniques for classifying GitHub issues. Our study evaluates the effectiveness of feature extraction methods using pretrained language models and the suitability of different classifiers for this task.

The key contributions of this work are as follows:

- Development and evaluation of a framework that combines feature extraction using contextual language models with various machine learning classifiers.
- Comparative analysis of multiple classification models, including traditional and deep learning-based approaches, on a benchmark dataset.
- Identification of optimal feature extraction techniques and classifiers for the automated triaging of GitHub issues.

This report is organized in the following way. Section II discusses the background and related works. Section III explains the technical approach and methodology used in the study. Section IV provides the implementation details of the project. Section V presents the empirical evaluation, including information on the dataset, performance metrics, results, discussions, and threats to validity. Finally, Sections VI and VII cover the lessons learned and conclude the paper, along with potential directions for future research.

II. Background and Related Work

Every software program encounters challenges when deployed in real-world scenarios. These challenges often manifest as bugs in the existing code, questions regarding the functionality of the software, or requests for new features. Such diverse issues necessitate a systematic approach to categorize them based on their context and priority, enabling efficient resolution and resource management. Issue Tracking Systems (ITSs) serve as a cornerstone for managing software maintenance and evolution, allowing developers to monitor, classify, and address various issues throughout the software development lifecycle.

ITSs, such as those integrated into platforms like GitHub, empower users to report bugs, request new features, or raise questions about projects. By providing detailed entries, ITSs

*All authors have given equal contribution to the work. The code implementation for this study can be found here.

enable software engineers to analyze the nature of each issue. For bug reports, in particular, these systems assist in pinpointing and narrowing down the files that require modifications, thereby streamlining the debugging process. However, given the dynamic nature of software projects and the public accessibility of platforms like GitHub, developers often face an overwhelming influx of issues. With hundreds of new entries generated daily for high-traffic repositories, manual triaging becomes increasingly infeasible. This process, although crucial, is prone to human error, time-intensive, and laborious.

Recent advancements in machine learning (ML) have brought transformative changes to the domains of software and requirements engineering [11]. In software engineering, ML algorithms are leveraged for tasks such as bug prediction, issue classification, code generation, and defect detection. These techniques enhance efficiency by automating repetitive tasks, thus reducing human effort and improving workflow consistency. Similarly, in requirements engineering, ML facilitates the elicitation, classification, and prioritization of requirements, ensuring alignment with project goals and stakeholder expectations.

The automation of issue classification is a particularly impactful application of ML in software engineering. By leveraging ML models, issues can be automatically categorized into labels such as "bug," "feature," or "question," which helps manage large datasets and accelerates project workflows. Traditional ML models such as Support Vector Machines (SVMs) and Random Forests have shown promising results in this domain. However, recent advancements in deep learning and transformer-based architectures, such as BERT (Bidirectional Encoder Representations from Transformers) and RoBERTa, have significantly enhanced issue classification. These models excel at capturing contextual nuances in textual data, making them particularly effective for understanding the diverse and often unstructured nature of issue descriptions.

Automated issue classification, therefore, represents a vital step toward optimizing project management in software development. By reducing manual triaging efforts and increasing classification accuracy, ML and deep learning techniques contribute to the development of more robust, scalable, and efficient software solutions. This paper explores these techniques, focusing on their application to GitHub issue classification and their potential to revolutionize issue tracking and management.

Many of the previous works have used Bidirectional Encoder Representations from Transformers (BERT) and its variants for the task of classifying issues. Siddiq et al. [12] fine-tuned a pre-trained BERT-base-uncased model on the NLBSE'22 training dataset which consists of 722,899 training samples and 80,518 testing samples. Another paper by Alam et al. [1] first fine-tuned the RoBERTa-Large model which is a variant of BERT on their preprocessed dataset and then performed hyper parameter tuning of their model to infer which set of parameters performed the best. Similarly, Ebrahim et al. [4] used a pre-trained RoBERTa transformer and trained an adapter and classification head simultaneously on the given dataset. Finally, they attach the adapter to the transformer and merge it with the classification head to classify the given issue. Nadeem et al. [9] used the transformer-based model RoBERTa-base for issue report classification in which they fine-tuned their model on a dataset of 55,000 issue reports. They optimize hyperparameters, including a learning rate,sequence length, batch size, and epochs and deploy the model as a GitHub-integrated application called Automatic Issue Classifier (AIC). Rejithkumar et al. [10] fine-tuned two variants of the T5-large model—T5-large and VMware/flan-T5-large-alpaca—on the issue report classification dataset by fine tuning hyperparameters such as epochs, learning rate, batch size, weight decay, target length, and max input length and observed that VMware/flan-T5-large-alpaca achieved the best results. Dissanayake et al. [3] compared Artificial Neural Network (ANN) and Long Short-Term Memory (LSTM) networks for Bug Priority Prediction using GloVe, TF-IDF and Word2Vec to generate feature vectors on a Bugzilla bug report dataset. Izadi et al. [6] trained several multi-label classifiers such as LR and DistilBERT-based models to recommend the top topics for different software repositories. Heo et al. [5] compared IssueBERT, CodeBERT, BERTOverflow, RoBERTa and seBERT for multilabel issue report classification, demonstrating that IssueBERT, pre-trained with domain-specific issue data, outperforms the other models, even though it is trained with a much smaller dataset.

III. Technical Approach

This section describes the proposed methodology for classifying issue reports. The overall framework is illustrated in Fig. 1. Initially, the dataset is downloaded from the NLBSE'24 competition page [7]. Following this, preprocessing steps are applied to the data. Embedding features are then extracted using different models, and are combined and normalized. The processed features are subsequently fed into five classifiers, namely SVM, RF, XGBOOST, LightGBM, LSTM and DistilBERT. The final step involves categorizing the reports as one of the categories.

A. Preprocessing

The preprocessing pipeline involves multiple steps to clean and standardize the textual data. First, any content enclosed within triple backticks (...) is removed to eliminate extraneous code or text blocks. Next, newline characters are replaced with spaces to ensure the text is formatted as a single line. Links, identified by standard URL patterns (e.g., starting with http or https), are also removed to eliminate irrelevant references. Numeric digits are then stripped from the text, followed by the removal of special characters, except for question marks, to retain meaningful punctuation. Consecutive whitespace characters are replaced with a single space to ensure consistent spacing. The cleaned and processed text is then stored for further analysis, ensuring that only relevant content is retained for downstream tasks.

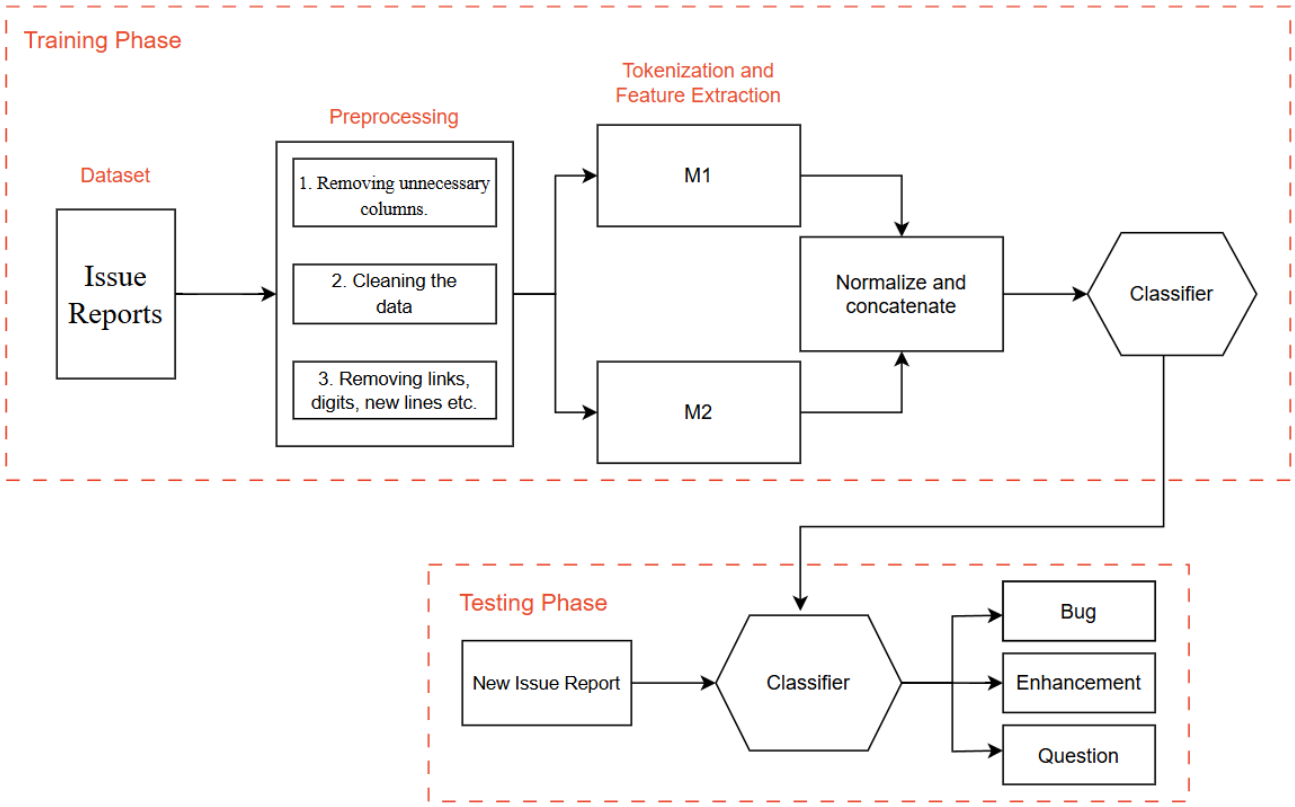

Fig. 1: An overview of the proposed approach

B. Feature Extraction

Following the preprocessing step, we employ two pretrained contextual models to extract textual features from the title and body field – RoBERTa and Sentence Transformers. Different combinations of these two models are used to extract the features. RoBERTa, an advanced transformer-based model developed by Facebook AI in 2019. It builds upon the BERT architecture, introducing enhancements in training methodology to improve performance across various natural language processing (NLP) tasks. Sentence Transformers are a type of model specifically designed for producing dense vector embeddings of sentences or text fragments. They extend pre-trained transformer models (like BERT or RoBERTa) to handle sentence-level or passage-level tasks effectively. In this study, the output from the penultimate layer of the RoBERTa model is utilized as the feature score for the summary field. For other fields, the Sentence Transformer model is applied to calculate the text embeddings. This results in embeddings for all the tokens corresponding to a given sample. The [CLS] embeddings are considered as it consolidates the summary of the sentences within one token embedding. Finally, the feature embeddings of the *Body* and *Title* features are normalized and concatenated and fused to create a feature matrix.

C. Classifier

To identify the most effective classifier for the proposed method, the extracted features are individually fed into five classifiers for training, and their performance is evaluated. The classifiers considered in this study include SVM, RF, XGBoost, LightGBM, LSTM and DistilBERT. Instead of fine-tuning the whole DistilBERT, adapters are attached to the model and trained for computational efficiency.

DistilBERT is a lightweight transformer model derived from BERT, designed to be smaller and faster while retaining a high level of performance. It achieves this through knowledge distillation, where a larger BERT model acts as a teacher, and DistilBERT learns a compressed version of the original model. Adapters are small, trainable neural modules added to pre-trained models like DistilBERT to enable fine-tuning for specific tasks without modifying the model's core parameters. This approach makes fine-tuning efficient, as only the adapter parameters are updated during training, preserving the original model's general knowledge. Combining DistilBERT with adapters is ideal for resource-constrained environments, providing a balance between computational efficiency and task-specific customization.

IV. Implementation

To implement our approach, we utilized the HuggingFace Transformers library to leverage pre-trained models. Specifically "all-mpnet-base-v2" variant of Sentence Transformers, and "FacebookAI/roberta-base" RoBERTa model were used for generating embeddings. Contextual models embeddings are a better fit for classifying issues than static model embeddings because of the nature of the prompts which are given by users in the Body and Title fields, which might relate different phrases within the texts. We refer to these embedding models as M1 and M2 for the purpose of simplicity. All four combinations of the two techniques are experimented with. Both M1 and M2 output a [CLS] token embedding as a vector of size [768,]. The extracted embeddings were subsequently normalized using the StandardScaler class from the scikit-learn library to ensure consistent feature scaling. Following normalization, the embeddings were concatenated to form a unified feature vector of size [1536,]. The combined feature vector was then passed into the classifier model, where performance evaluation was conducted using tools the different classification models. SVC and RF are implemented using the scikit-learn library. Additionally, for deep learning components such as the Long Short-Term Memory (LSTM) network, optimizer configuration, and integration with HuggingFace adapters, we employed PyTorch as the primary framework. This implementation allowed us to effectively utilize pre-trained models for feature extraction and seamlessly integrate machine learning and deep learning components to evaluate the classification performance of our methodology. Hyperparameter tuning using grid search is done for all the models except the pre-trained DistilBERT to choose the best model configuration.

V. Empirical Evaluation

A. Research Questions

- *What machine learning and deep learning models are best suited for issue classification?*
  The study compares traditional models like SVM, RF, XGBoost, and LightGBM with deep learning models like LSTM and DistilBERT.
- *How does preprocessing affect the quality of issue classification?*
  Preprocessing steps, including the removal of unnecessary text (e.g., code snippets, URLs), standardization, and normalization, play a crucial role in enhancing dataset quality. These steps ensure that only relevant information is retained, reducing noise and improving the performance of feature extraction models and classifiers.
- *Are all features relevant to our problem?*
  Columns like repo_name and date_created don't hold much importance in predicting the label for the specific issue and would add random noise to the data.
- *What trade-offs exist between computational efficiency and classification performance?*
  The study highlights a trade-off between computational cost and model accuracy: LightGBM offers computational efficiency but sacrifices some accuracy, while DistilBERT provides near-perfect accuracy but demands higher computational resources. Balancing these trade-offs depends on the application's resource constraints and performance requirements.

B. Description of Dataset

The dataset used for this study consists of 3,000 GitHub issue reports, with an example issue report structure shown in Table 1. The dataset includes the following columns:
Repo: The GitHub repository from which the issue was taken.
Created at: The date and time when the issue was posted to the repository.
Title: A concise summary of the issue report.
Body: A detailed description that may include reproduction steps, logs, or contextual information.
Label: The class of the issue report, categorized as either a bug, a feature, or a question.

The issue reports are sourced from five diverse and widely used repositories: facebook/react, tensorflow/tensorflow, microsoft/vscode, bitcoin/bitcoin, and opencv/opencv, representing a broad range of domains such as web development, machine learning, code editing, cryptocurrency, and computer vision. The 3,000 reports are evenly distributed across the repositories, with 600 reports per repository. Each repository contributes 300 training reports and 300 test reports, divided equally among the three categories: bug, feature, and question. This results in 100 training reports and 100 test reports per category per repository.

TABLE I: A sample from the dataset

repo	opencv/opencv
created at	15-01-2022 02:39
label	feature
title	Reading BigTiff images
body	**Merge with extra: https://github.com/opencv/opencv_extra/pull/952** resolves #18717. See details at https://github.com/opencv/opencv/wiki/How_to_contribute#making-a-good-pull-request - [ ] I agree to contribute to the project under Apache 2 License. - [ ] To the best of my knowledge, the proposed patch is not based on a code under GPL or other license that is incompatible with OpenCV. - [ ] The PR is proposed to the proper branch.

C. Analysis Procedure and Metrics

We evaluate the performance of all the models with each embedding combination using the following metrics:

Precision (P): Precision measures the proportion of correctly predicted labels to the total predicted observations for a given class. It is defined as:

$$P = \frac{TP}{TP + FP}$$

where TP (true positives) represents the number of correctly predicted records, and FP (false positives) refers to records where the label was incorrectly predicted.

Recall (R): Recall quantifies the proportion of correctly predicted observations in a class to the total number of observations in that class. It is calculated as:

$$R = \frac{TP}{TP + FN}$$

where FN (false negatives) denotes the observations in a given class that were incorrectly predicted as belonging to other classes.

F1-Score (F1): The F1-score provides a balanced measure by combining precision and recall into a single metric using their harmonic mean, defined as:

$$F1 = 2 \times \frac{P \times R}{P + R}$$

For each class, we compute Precision, Recall, and F1-Score.

Accuracy (A): Accuracy measures the overall correctness of the model by calculating the proportion of correctly classified observations to the total number of observations. It is defined as:

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

where TN (true negatives) represents the number of correctly identified negative records.

D. Results and Discussion

The experiment involved embeddings from Sentence Transformers, RoBERTa, and DistilBERT models applied to the *Body* and *Title* features of issues. These embeddings were evaluated using various models: SVC, Random Forest (RF), XGBoost, LightGBM, LSTM, and DistilBERT. Performance was measured using F1-score, Accuracy, Precision, and Recall metrics.

Using Sentence Transformer embeddings, SVC achieved a highest F1-score of 0.776, showcasing its efficiency with high-dimensional contextual features. RF exhibited consistent but moderate performance, with a highest F1-score of 0.722, indicating limitations in handling complex embeddings. XGBoost demonstrated a slight improvement over RF, achieving an F1-score of 0.732. LightGBM's results (F1-score: 0.754) highlight its capability to efficiently process contextual embeddings with satisfactory results. The LSTM model achieved a similar result with a highest F1-score of 0.747. DistilBERT recorded a remarkable F1-score of 0.992, with equally high values for Accuracy (0.992), Precision (0.992), and Recall (0.992). This demonstrates the potential of transformer-based fine-tuned models with adapters to provide near-perfect issue classification.

DistilBERT outperformed all other models by a significant margin, indicating the efficacy of leveraging pre-trained transformer-based architectures in combination with task-specific embeddings. Traditional models like RF and XGBoost were outperformed by SVC, suggesting the latter's compatibility with high-dimensional embeddings. The experiments also highlight the critical role of choosing the right combination of embedding generation techniques in the performance of classification models. The combination of RoBERTa for *Body* and Sentence-Transformer for *Title* yields overall better results across all metrics as compared to its counterpart combinations. The superior performance of DistilBERT aligns with the trend of transformer-based architectures dominating NLP tasks. Conversely, the relatively lower performance of RF and XGBoost indicates the challenges faced by tree-based ensemble methods in handling contextual embeddings. LightGBM, while computationally efficient, demonstrated a trade-off in performance compared to transformer-based models.

E. Threats to Validity

In this section, we review the various threats to the validity of our study based on four groups of internal, external, construct, and conclusion validity.

Internal Validity The choice of Sentence Transformers, RoBERTa, and DistilBERT embeddings may influence the performance outcomes. Alternative embedding models, such as GPT-based embeddings, were not included, which might yield different results. Also, limited hyperparameter optimization was conducted for the machine learning models, which might restrict their performance potential. However, all chosen hyperparameters for grid search are in ranges which usually have been observed to give better results. The relatively smaller size of the dataset might also lead to sampling bias, potentially leading to overfitting or under-generalization.

External Validity The findings rely heavily on the specific dataset used. Performance may vary when the models are applied to other datasets with different issue characteristics, structures, or distributions. The embeddings and models were not fine-tuned for a specific domain. As such, the results may not generalize to domain-specific tasks without further adaptation.

Construct Validity We have used standard metrics like F1-score, Accuracy, Precision, and Recall, which are proven to be ideal for classification evaluation in academia and research. The results indicate the employed approach has been successful in comparing different models and embedding techniques for issue classification.

Conclusion Validity While several models were evaluated, other potential classifiers (e.g., advanced deep learning architectures like GPT or BERT-Large) were not tested. This might limit the scope of conclusions regarding optimal approaches for issue classification. However, fine-tuning transformers and LLMs require high computational resources, which is a drawback of such approaches.

VI. Lessons Learned

Through the course of this research, several valuable insights emerged. The choice of embeddings significantly impacts classification performance. Advanced pre-trained models like DistilBERT provided near-perfect results, emphasizing the importance of using state-of-the-art models for complex tasks. While DistilBERT+Adapter model outperformed traditional machine learning models, simpler classifiers like SVC demonstrated competitive performance in comparison with other traditional methods, when paired with high-quality

TABLE II: Results of experiments run with different embedding techniques and machine learning models

Embedding Techniques		Model	Metrics			
Body	Title		F1	Accuracy	Precision	Recall
Sentence Transformer	Sentence Transformer	SVC	0.765	0.765	0.765	0.765
		RF	0.706	0.706	0.706	0.706
		XGBOOST	0.712	0.712	0.712	0.712
		LIGHTGBM	0.734	0.734	0.734	0.734
		LSTM	0.722	0.723	0.733	0.723
		DistilBERT	0.992	0.992	0.992	0.992
Sentence Transformer	RoBERTa	SVC	0.749	0.749	0.749	0.749
		RF	0.704	0.704	0.704	0.704
		XGBOOST	0.712	0.712	0.712	0.712
		LIGHTGBM	0.734	0.734	0.734	0.734
		LSTM	0.727	0.732	0.732	0.732
		DistilBERT	0.991	0.991	0.991	0.991
RoBERTa	Sentence Transformer	SVC	0.776	0.776	0.776	0.776
		RF	0.722	0.722	0.722	0.722
		XGBOOST	0.732	0.732	0.732	0.732
		LIGHTGBM	0.754	0.754	0.754	0.754
		LSTM	0.747	0.748	0.748	0.748
		DistilBERT	0.992	0.992	0.992	0.992
RoBERTa	RoBERTa	SVC	0.758	0.758	0.758	0.758
		RF	0.703	0.703	0.703	0.703
		XGBOOST	0.721	0.721	0.721	0.721
		LIGHTGBM	0.734	0.734	0.734	0.734
		LSTM	0.738	0.737	0.739	0.737
		DistilBERT	0.991	0.991	0.991	0.991

embeddings. This suggests that model complexity should be chosen based on computational resources, to find the sweet spot between low computation and high accuracy. A balance between computational efficiency and performance must be considered. LightGBM, being more computationally efficient, was outperformed by DistilBERT, indicating a trade-off between speed and accuracy.

Fine-tuning embedding models on task-specific data remains a critical step. The current results suggest that even slight adjustments to embeddings or models can drastically affect performance. Also, the use of adapters in fine-tuning the much more complex DistilBERT leads to benchmark performance in a much shorter time period and with limited resources, as compared to fine-tuning the entire model. This highlights the importance of the base-model weights being frozen while training the light adapters, which can be used with other transformer-based models such as T5, GPT etc.

VII. Conclusion

This research explored issue classification using a combination of advanced embedding techniques and machine learning models. The findings demonstrate that the transformer-based model, DistilBERT, along with the embedding pair of RoBERTa for the body feature and Sentence-Transformer for the title feature achieves a highest F1-score of 0.992. Among the models evaluated, DistilBERT consistently outperformed traditional classifiers like Random Forest and XGBoost. Within traditional models, SVC outperformed its classifier counterparts.

Future work should focus on fine-tuning embedding models, utilizing other state-of-the-art NLP models and LLMs, and evaluating these models on larger, more diverse datasets with more features and samples. This study serves as a foundation for further research into embedding-driven issue classification and its applications across varied domains. Future efforts should also prioritize dataset enrichment to better capture real-world variability.

REFERENCES

[1] Khubaib Amjad Alam, Ashish Jumani, Harris Aamir, and Muhammad Uzair. Classifai: Automating issue reports classification using pre-trained bert (bidirectional encoder representations from transformers) models. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*, pages 49–52, 2024.

[2] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] DNA Dissanayake, RAHM Rupasingha, and BTGS Kumara. Automatic bug priority prediction using lstm and ann approaches during software development. 2024.

[4] Fahad Ebrahim and Mike Joy. Few-shot issue report classification with adapters. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*, pages 41–44, 2024.

[5] Jueun Heo, Gibeom Kwon, Changwon Kwak, and Seonah Lee. A comparison of pretrained models for classifying issue reports. *IEEE Access*, 2024.

[6] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering*, 26(5):93, 2021.

[7] Rafael Kallis, Giuseppe Colavito, Ali Al-Kaswan, Luca Pascarella, Oscar Chaparro, and Pooja Rani. The nlbse'24 tool competition. In *Proceedings of the Third ACM/IEEE International Workshop on NL-based Software Engineering*, pages 33–40, 2024.

[8] Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364, 2019.

[9] Anas Nadeem, Muhammad Usman Sarwar, and Muhammad Zubair Malik. Automatic issue classifier: A transfer learning framework for classifying issue reports. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, page 421–426. IEEE, October 2021.

[10] Gokul Rejithkumar, Preethu Rose Anish, and Smita Ghaisas. Text-to-text generation for issue report classification. In *Proceedings of the Third ACM/IEEE International Workshop on NL-Based Software Engineering*, NLBSE '24, page 53–56, New York, NY, USA, 2024. Association for Computing Machinery.

[11] Saad Shafiq, Atif Mashkoor, Christoph Mayr-Dorn, and Alexander Egyed. A literature review of using machine learning in software development life cycle stages. *IEEE Access*, 9:140896–140920, 2021.

[12] Mohammed Latif Siddiq and Joanna CS Santos. Bert-based github issue report classification. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, pages 33–36, 2022.