Automatic Issue Classifier: A Transfer Learning Framework for Classifying Issue Reports

Anas Nadeem
Dept. of Computer Science
North Dakota State University
Fargo, USA
anas.nadeem@ndsu.edu

Muhammad Usman Sarwar
Dept. of Computer Science
North Dakota State University
Fargo, USA
muhammad.sarwar@ndsu.edu

Muhammad Zubair Malik
Dept. of Computer Science
North Dakota State University
Fargo, USA
zubair.malik@ndsu.edu

***Abstract*—Issue tracking systems are used in the software industry for the facilitation of maintenance activities that keep the software robust and up to date with ever-changing industry requirements. Usually, users report issues that can be categorized into different labels such as bug reports, enhancement requests, and questions related to the software. Most of the issue tracking systems make the labelling of these issue reports optional for the issue submitter, which leads to a large number of unlabeled issue reports. In this paper, we present a state-of-the-art method to classify the issue reports into their respective categories i.e. bug, enhancement, and question. This is a challenging task because of the common use of informal language in the issue reports. Existing studies use traditional natural language processing approaches adopting key-word based features, which fail to incorporate the contextual relationship between words and therefore result in a high rate of false positives and false negatives. Moreover, previous works utilize a uni-label approach to classify the issue reports however, in reality, an issue-submitter can tag one issue report with more than one label at a time. This paper presents our approach to classify the issue reports in a multi-label setting. We use an off-the-shelf neural network called RoBERTa and fine-tune it to classify the issue reports. We validate our approach on issue reports belonging to numerous industrial projects from GitHub. We were able to achieve promising F-1 scores of 81%, 74%, and 80% for bug reports, enhancements, and questions, respectively. We also develop an industry tool called Automatic Issue Classifier (AIC), which automatically assigns labels to newly reported issues on GitHub repositories with high accuracy.**

***Index Terms*—software maintenance, software quality, GitHub issue report classification**

I. Introduction

Software maintenance refers to the changes made to the software in order to make it more robust, efficient, and to make it adaptable to a modified environment [1]. Issue tracking systems are one of the most critical means for the maintainers to enable rigorous yet effective software evolution tasks. Issue tracking systems allow the software developers to report, manage and keep track of the tickets [2]. They are capable of being the single point of reference for nearly all the maintenance activities such as resolving reported bugs, making the project more efficient, and adding new features.

GitHub is a version control and source code management service, which empowers the users in several ways such as to host the source code, keeping track of versions, and release and maintain the software. It is the largest source code hosting platform with more than 50 million developers and more than 100 million public repositories [1] as of the year 2018. Similar to the other conventional bug-tracking systems such as Jira[2], MantisBT[3], and Zoho[4], GitHub also provides an integrated issue tracking system. Using the GitHub issue tracking system, a user can interact with the maintainers of the repository. For instance, any user who wants to report a bug, request a new feature, or wants to ask questions related to the software might make use of the issue tracking system.

To create an issue report [5] in the GitHub issue tracker, users are required to provide a title and a brief description as the body. However, the labels associated with the issue are optional. Issue submitter can choose these labels from the list of default issue labels such as 'bug', 'enhancement', 'documentation', and 'question' or it can be custom defined by the issue submitter. Additionally, the issue submitter can tag the issue report with multiple labels at a time. Labelling these issues enables the project team to prioritize these issues based on the label assigned. For example, while the software is close to a release, it might be important to make the software more robust instead of adding new features. While all these categorized issue reports help the software maintainers to identify, reproduce and finally make appropriate changes in the software, it also complicates the tasks for the software maintainer due to the inconsistent nature of labels. For instance, the bug-report label can be written in multiple ways such as 'bug', 'there is a bug' etc., which makes this a challenging task. Usually, software project repositories either rely on the issue-submitter to assign a label to the issue report based on their intuition or they rely on the software development team to manually classify these issues to their respective categories. The manual classification approach works well for repositories in which issues are reported with a low frequency. However, there are numerous popular repositories on GitHub which have a very large number of issues reported every day, making it a challenging task to manually label them. Also, labelling these

[1]https://venturebeat.com/2018/11/08/github-passes-100-million-repositories/
[2]https://www.atlassian.com/software/jira
[3]https://www.mantisbt.org/
[4]https://www.zoho.com/bugtracker/
[5]Terms 'issue' and 'issue report' are used interchangeably throughout the paper.

issues on such repositories is a non-trivial task and generally requires specific domain knowledge and a thorough analysis of the body of the issue.

Classification of GitHub issue reports has recently been of great interest as an enabling path for a more involved GitHub-based data analysis. However, the existing approaches to solve the issue report classification problem do not use advanced language modeling and rely on keyword-based approaches [2]–[6]. Traditional, keyword-based approaches suffer from a high false-positive and a high false-negative rate. This is because they view the problem as bag-of-words and are not aware of the full language model, hence they do not capture contextual relationships among the words [7]. Also, the existing studies view the problem as a multi-class classification problem that assumes an issue can be only associated with a single label. However, an issue submitter can tag the issue label with multiple labels at the same time. Therefore, we have to treat the issue classification problem as multi-label classification. In multi-label classification settings, an issue report can be associated with more than one label at a time.

To address the aforementioned problems, we used a transformer-based model called RoBERTa [8]. RoBERTa is an optimized variant of BERT, which uses improved training methodology and larger data to achieve better results than BERT [9]. RoBERTa has the ability to consider the contextual relationship between the word while making the prediction. To train RoBERTa, we take the GitHub issue data set labeled with the 3 labels i.e. 'Bug', 'Enhancement', 'Question'. Here, we only used issues that are tagged with the default GitHub labels. As we are essentially performing transfer learning, we need significantly fewer data samples to build an accurate model as compared to millions of data examples required to train a model from scratch [7]. We evaluated our approach on approximately 55,000 randomly sampled issue reports associated with the top 200 repositories across 55 popular languages. We were able to achieve promising results with an F1-Score nearing 80% with significantly fewer training examples.

In our work, we tried to answer the following research questions:

1) RQ1: Does the transformer-based approach out-perform the traditional keyword-based approaches at classifying GitHub issue reports?
2) RQ2: Does the transformer-based model have the ability to be incorporated into an industry-level tool?

This work has immediate benefits to the industry as it enables efficient tracking of issue reports on GitHub. First, it can help the software development team track the software maintenance process effectively. Secondly, the categorized issue classification can help in effective development documentation that can help the project team to achieve effective software maintenance. Thirdly, having accurately categorized issue reports would help in routing the issue to the right software developer, who is working on the specific module. Lastly, categorized issue reports can help the project manager to optimize the resource allocation process, if a particular module is getting a lot of bug reports that implies the need of assigning resources for the purpose of improving the associated code-base.

The following is a summary of our contributions:

- We present a transformer-based approach for classifying the GitHub issue reports in a multi-label setting.
- We conducted our evaluation on GitHub issue-tracker dataset across 55 different programming languages.
- We develop and release a tool, Automatic Issue Classifier (AIC) on top of our transfer-based approach. AIC is capable of being deployed in industrial-scale software repositories to automatically classifies issue reports as soon as they are reported.

This paper is organized as follows: Section II discusses the previous relevant studies. Section III discusses the methodology we used for the data collection, and issue report categorization. Section IV discusses the evaluation of our approach. Section V discusses the implication of these results in the industry. Section VI discusses the threats to validate the study. Section VII discusses the potential future avenues. Finally, section IX concludes the study.

II. Related Work

Previous studies in the domain of classification of issues on issue-trackers range from the binary classification of bugs/non-bugs to the categorization of issues using multi-class classification methods. Several research works have proposed methods ranging from using keyword-based approaches to sophisticated machine learning models. We discuss these studies and their limitations in order to lay the grounds for our research.

Antoniol et al. [4] proposed a key-word based approach in order to distinguish bug issues from non-bug issues. They manually labeled 1,800 issues associated with repositories of Mozilla, Eclipse, and JBoss and further fed these issues to Alternating Decision Trees, Naive Bayes, and Logistic Regression-based models resulting in up to 82% correct results. Our work proposes a multi-label method to classify these issues into three mutually exclusive labels. Additionally, we evaluate our results on a larger set of issues from a diverse set of repositories.

Kallis et al. [2] presented TicketTagger, a tool to categorize GitHub issues. They used a fastText [10] based model to classify the GitHub issue into bugs, enhancement, and questions categories. They were able to achieve an F1-score of 82% over all three of the categories. They treated the issue classification problem as a multi-class problem where one issue can belong to a single label at a time. Our work used the same labels as used by Kallis et al. [2]. However, we solve this problem as a multi-label classification problem, where an issue report can have multiple labels at the same time.

Fan et al. [6] proposed a two-stage framework for classifying the issue reports. They perform their analysis on over 252,000 issue reports from 80 popular GitHub projects. Further, they compared their results with four traditional machine learning methods including Naive Bayes, Logistic Regression, Random Forest Tree, Support Vector Machine. Our work,

in contrast, leverages the power of pre-trained transformer-based models which produce state-of-the-art results without requiring a large number of training examples.

Chawla et al. [3] proposed an automated approach to categorize the type of issue reports. They utilized a Fuzzy Logic based system to classify the issue types and evaluated their results on issue reports from HTTPClient, Jackrabbit, and Lucene. They were able to achieve an F-1 score of 0.83, 0.79, 0.84 for the three repositories, respectively. However, they evaluated their model on results on a limited number of well-maintained repositories which might be misleading.

Fazayeli et al. [5] used text-based classification to label GitHub issues. However, they only evaluated their framework for only a single repository i.e. 'git-for-windows'. Also, they treated the problem as a binary classification problem with bug and non-bug labels. However, our framework categorically attempts to assign multiple labels to a GitHub issue report. Also, we evaluated our approach on a diverse data set with projects from various programming languages.

The above-discussed studies utilized keyword-based features to categorize the issue reports, which suffer from a high rate of false positives. The primary reason of which is that they do not consider the contextual relationships between words while classification. Also, these studies treat the issue classification problem as a multi-class classification problem. However, in real-world scenarios, an issue can be associated with more than one label at a time and should be addressed as a multi-label classification problem.

Attention-based networks derived from human intuition have resulted in significant improvement in various natural language processing tasks [11]. Attention-based networks are capable of focusing on context-relevant details in a given text while ignoring irrelevant keywords. Bahdanau et al. [12] were the first to introduce an attention mechanism for the purpose of machine translation, commonly referred to as additive attention. Recent advancements in NLP has led to wide adoption of self-attention based Transformers [13] models. Pre-trained transformers are trained on a large natural language corpus and further, they can be fine-tuned for the downstream NLP tasks such as machine translation, and text classification.

In recent work, Devlin et al. [14] introduced Bidirectional Transformers for Language Understanding (BERT) which leverages the Transformer-architecture. BERT was originally pre-trained on BooksCorpus containing 800M words and English Wikipedia containing 2,500 million words. This pre-training over large data sets enables BERT to get familiar with language vocabulary. Further fine-tuning the pre-trained model for specific tasks refers to Transfer Learning. There are several BERT variants based on architecture such as the number of input layers, hidden layers, self-attention heads, and parameters used. While BERT is a powerful framework for NLP tasks, tweaking it can substantially improve its performance. Liu et al. [8] proposed RoBERTa which is a robust and optimized variant of BERT. They proposed several design changes such as removing the next sequence prediction objective, changing the masking pattern based on the data itself. They show that these changes along with pre-training the model for a longer duration and in larger batches results in substantial improvement and give state-of-the-art results on benchmark data sets such as GLUE, SQuAD, and RACE datasets.

III. Methodology

This section presents our proposed approach towards the classification of issue reports on GitHub. Our approach consists of the following phases: (1) Data Collection phase, which involves data collection from GitHub (2) Data Pre-processing phase, which involves data transformation and cleaning tasks. (3) Model Construction phase, which involves training a multi-label transformer-based model i.e. RoBERTa. (3) Automatic Issue Classifier (AIC), a GitHub application which can be leveraged to classify GitHub issues-reports on any GitHub repository. Figure 1 shows the overview of our approach. We briefly highlight the different phases of our approach in this section and explain details of each phase in the subsequent subsections.

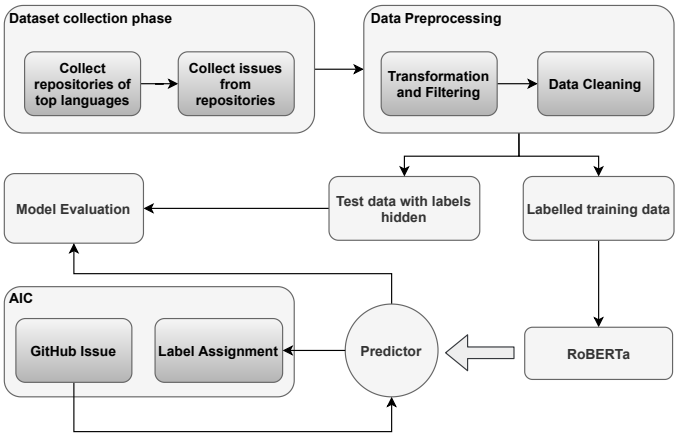

Fig. 1. Overview of the Framework: Starts with data collection phase. Further, gathered dataset is pre-processed. The pre-processed data is then splited into train and test sets in order to evaluate the results.

A. Data Collection

For this work, we used GitHub as our data source. First, we identified the top 55 candidate popular languages using the IEEE Spectrum Top Programming Languages list [15]. Further, we fetched the top 200 popular repositories across each programming language. Each repository on GitHub has an associated star count, which represents the popularity of the repository [16]. Here, we identified the project by their primary language. The primary language of the project is defined as the one in which the largest number of source code files are written [17]. Lastly, we fetched issues associated with each repository along with the related attributes. These attributes include, but are not limited to; title, body, and labels of issues.

Our data set contains issues from numerous open-source and industrial projects such as Laravel, Kubernetes, Linux, Spark. Table I shows the top three projects from our data across some

Language	Projects
Assembly	Apollo-11, MS-DOS, mal
C	linux, netdata, redis
C++	tensorflow, electron, terminal
PHP	laravel, jQuery-File-Upload, SecLists
JavaScript	freeCodeCamp, vue, javascript
Go	go, kubernetes, awesome-go
Ruby	rails, jekyll, discourse
Dart	flutter, awesome-flutter, flutter-go
C#	shadowsocks-windows, PowerToys, PowerShell
Racket	pollen, hackett, frog
Objective-C	AFNetworking, SDWebImage, GPUImage
Scala	spark, prisma1, scala

TABLE I
TOP 3 GITHUB REPOSITORIES ASSOCIATED ACROSS DIFFERENT LANGUAGES.

Title	Body	Categories
USBhost: additional functions for keyboard appreciated	for USBhost it would be fine to have additional functions to read the USB keyboard:kbhit() getch() getche() getchar() gets() scanf()	bug, enhancement

TABLE II
ISSUE REPORT THAT IS LABELLED WITH BOTH BUG AND ENHANCEMENT CATEGORIES.

of the programming languages. In total, we were able to fetch 1,166,107 issues out of which 509,090 of the issues were unlabelled and 657,017 issues were labeled. The substantial amount of unlabelled issues provides us the opportunity to further dig down to this problem and provide an effective approach to categorize the issue reports. We used GitHub Search and Data API [6] to fetch this dataset. The complete dataset is available for public use [7].

B. Data Pre-processing and Transformation

Transformation and Filtering : We extracted the title and body of the issue and concatenated them together. We filtered our GitHub issue reports into three categories i.e. bug-report, enhancement, and question consistent with the prior work by Kallis et al. [2]. Here, each label is non-exclusive such that an issue can belong to more than one category. For instance, an issue can be labeled as a bug report and enhancement at the same time. Table II shows one such issue example.

The labels under consideration in our research are the default GitHub labels:

1) **Bug-report:** Issue that reports a bug or fault in the project.
2) **Enhancement:** Issue that requests the enhancement of the project, such as performance improvement, feature request.
3) **Question:** Issue that asks the question from the repository owner or maintainer.

We further filtered out the issues that were not in the English language and only used issues that are written in English. We used langdetect [8] to tag every issue with the language used in the text and found that 647,438 of the labeled issues were written in English while only 9,579 issues were written in other languages.

Pre-processing: We did not perform traditional NLP pre-processing steps such as stop-word removal, stemming, and lemmatization. The RoBERTa has its own tokenizer that does not require traditional NLP pre-processing steps to be applied to the text. However, we converted the text of all issue reports to lower case, as we are using RoBERTa with its base pre-trained model which is a case-sensitive model. Additionally, as our data set contains real-world issue reports, our data set has a class-imbalance problem. Less than 4% issues were labeled as questions. To resolve, the class imbalance problem, we randomly over-sampled the issue reports with less representative labels i.e. question label.

C. Model Construction

Pre-trained transformer-based models are trained on a large natural language corpus, which empowers them to extract the detailed contextual word representation from the text. We used one such transformer-based model called RoBERTa [8], a robust and optimized variant of BERT [9]. The reason behind choosing RoBERTa over BERT is its state-of-the-art results on benchmark data sets such as GLUE, RACE, and SQuAD with over 2-20% improvement over BERT. RoBERTa has several configurations which can be tuned on the basis of the pretraining approach. We use 'roberta-base' which consists of 12 transform layers, 768 hidden layers, and 12 self-attention heads. We used simpletransformers [9] to fine-tune our model. The package uses HuggingFace's [10] implementation of the model.

Model Training: As we are using a pre-trained model, which essentially requires less number of training examples to be fine-tuned as compared to model trained from scratch, we randomly sampled 55,000 (approx) issue reports. We then divided the issue data set into 80%-20% train-test split and fed the training data set to fine-tune our model. We trained our model for 5 epochs with a learning rate of 4e-05, a maximum sequence length of 128, and a batch size of 8. We trained our model using Google Colab [11]. With 12 GB RAM, Tesla T4 14 GB GPU, and 2 Intel Xeon CPUs @ 2.2GHz it took an hour to train the model.

D. Automatic Issue Classifier

We deployed our model using a GitHub application called Automatic Issue Classifier (AIC). Figure 2 explains the working flow of the application. AIC application can be integrated into any GitHub repository and can automatically assign the newly created issue report with its respective label. AIC is a Flask [12] web framework based application with a python runtime and is deployed on a WSGI server [13]. AIC application

[6]https://docs.github.com/en/rest
[7]https://www.kaggle.com/ansnadeem/github-issues
[8]https://github.com/Mimino666/langdetect
[9]https://github.com/ThilinaRajapakse/simpletransformers/
[10]https://huggingface.co/
[11]https://colab.research.google.com/
[12]https://flask.palletsprojects.com/en/2.0.x/
[13]https://wsgi.readthedocs.io/en/latest/index.html

can be integrated with any existing GitHub repository by accessing it through the application page [14]. After landing on the application page, click on the 'Install' button and select the GitHub repository you want to associate it with.

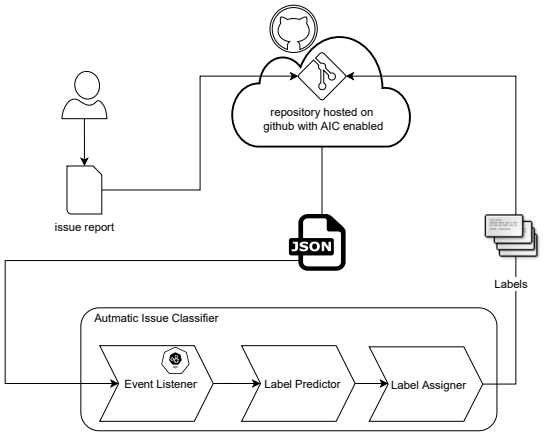

Fig. 2. AIC sits idle until a user creates an issue on a repository integrated with AIC. A new issue sends an event to AIC API, which in results predicts the issues based on the text in the issue report and assigns the label to it.

IV. Evaluation

In this section, we evaluate our proposed approach using GitHub issue reports data set and answer the research questions presented earlier in section I. Following metrics are used to evaluate the proposed approach:.

- **Precision** is the ratio of true positives to the sum of true positives and false positives. A precision value closer to 1 is the most desirable.
- **Recall** is a measure of how well we were able to identify relevant instances. It is defined as the ratio of true positives to the sum of true positives and false positives. A recall value of 1 indicates the best performance.
- **F1-Score** is defined as the harmonic mean of precision and recall. F-1 score closer to 1 is the most preferable.
- **Hamming Loss** is a commonly reported metric for multi-label classification. It is the fraction of incorrectly predicted labeled out of the total number of labels. A hamming loss value of 0 indicates the best results.

RQ1: Does the transformer-based approach perform better than the traditional keyword-based approach at classifying GitHub issue reports?

Table III shows the results of our approach over GitHub issue reports classification. Our RoBERTa based approach is able to predict results for bugs, enhancement, and questions category with a hamming loss of 0.17, 0.16, and 0.16 respectively. Similarly, we were able to achieve an F1 score of 81%, 74%, and 80% for bugs, enhancement, and questions category respectively which are comparable to results of existing studies. For instance, Kallis et al. [2] reported F1 scores of 83.1%, 82.3%, 82.5% for bugs, enhancement, and questions category respectively. It is observed that our results are slightly degraded than the results achieved by Kallis et al. [2]. This is expected because our framework proposes a multi-label approach as opposed to the multi-class approach. Moreover, they sampled data in a class if any assigned label contains the text 'question', 'bug' or 'enhancement' however label text can be 'not-a-bug' which in this case would also be incorrectly sampled as a bug report, our sampling, in contrast, is based on exactly matching these labels resulting in more accurate classification.

Category	Precision	Recall	F1-Score	Hamming Loss
Bug	81%	81%	81%	0.14
Enhancement	78%	72%	74%	0.15
Question	79%	81%	80%	0.15
Macro-Average	79%	78%	78%	0.15

TABLE III
Evaluation results of our proposed approach

RQ2: Does the transformer-based model have the ability to be incorporated into an industry-level tool?

We successfully adopted and deployed the transformer-based model in an industry tool called Automatic Issue Classifier (AIC). Figure 3 shows a demonstration of AIC automatically assigning a label to an issue report. When a user creates an issue, AIC automatically assigns it a label based on the context. We deployed the tool as a GitHub application to ensure easy integration with industry projects. To enable users to tailor the application according to specific requirements, the complete source code of our application is available via a GitHub repository[15].

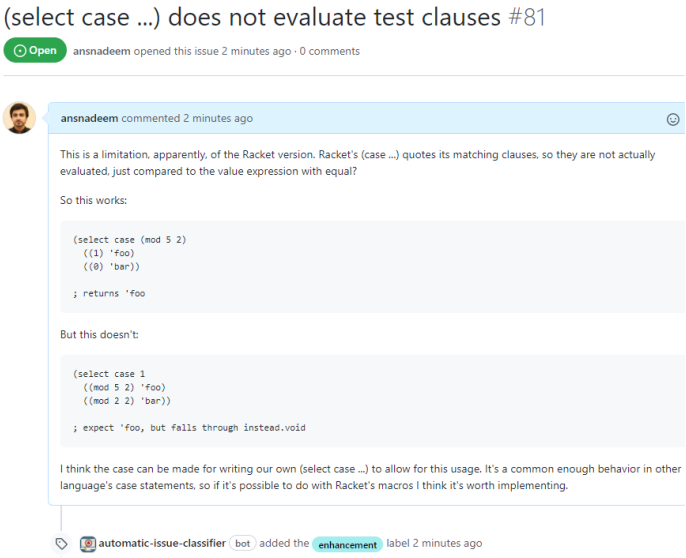

Fig. 3. Automatic Issue Classifier automatically assigns label to a newly reported issue on a repository.

V. Relevance to the Industry

Effective classification of issue reports can have an immediate effect on software development processes. First, it

[14]https://github.com/apps/automatic-issue-classifier

[15]https://github.com/ansnadeem/aic

can enable easy tracking of issue reports on GitHub, which could result in effective software maintenance. Secondly, the volume of each categorized issue report can help the project manager to review the project maintenance process at any point and make appropriate decisions. For instance, a large number of bug reports might indicate that the quality assurance process might need some help. Thirdly, having accurately categorized issue reports would help in routing the issue to the right software developer, who is working on the specific module/component. Also, accurately categorized issue reports can help the team leader to prioritize the tasks, and the release manager can figure out the way to stabilize the project. Lastly, the projects will be managed more efficiently and effectively, and ultimately would result in better products and projects.

VI. Threats to Validity

This section highlights some threats in validating our work.

- **Popularity Bias:** We are fetching results from the top repositories of the popular programming languages, so our results might be influenced by the popularity bias.
- **Language Imbalance Bias:** Our data set implies that some languages on GitHub are more popular than the rest. For instance, our data set contains a significant portion of the issues-reports that are associated with C# repositories.

VII. Future Work

Our future work would revolve around improving our industry tool:

- **Automatically Assigning Issues** We plan to extend our tool by making it capable of not only assigning labels but also assigning them to a developer.
- **More Categories:** We also plan to add more categories such as 'documentation', 'wontfix', and 'help wanted', 'duplicate' in the issue data set, and analyze if our model can also categorize them with similar accuracy.

VIII. Conclusion

We proposed a transformer-based approach towards the classification of GitHub issues and assigning them respective labels. Our model is able to well understand the context, hence provides accurate and efficient issue report classification. Our framework takes into account the non-exclusiveness of the issue-reports data and is capable of assigning multiple labels to issue reports at the same time. We extracted our data-set from GitHub repositories across the top 55 popular programming languages to ensure that our results and model are capable of generalizing across multiple languages. We evaluated our results over a data set containing three labels i.e., bugs, enhancement, and questions, and were able to achieve an F-1 score of 81%, 74%, and 80% respectively, which is comparable to existing issue classification results. We leveraged our state-of-the-art approach to develop an industry tool i.e. Automatic Issue Classifier that is capable of automatically labelling the issue reports. We believe that our tool will benefit the software engineering industry by enabling the teams to keep their focus on the necessary tasks by automating the manual task of labelling the issue reports.

IX. Acknowledgements

This work was supported by North Dakota State University College of Engineering. We would also like to thank Giorgos Karantonis[16] for sharing his key insights regarding the research problem.

References

[1] S. E. S. Committee et al., "Ieee standard for software maintenance," IEEE Std, pp. 1219–1998, 1998.

[2] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Predicting issue types on github," Science of Computer Programming, vol. 205, p. 102598, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642320302069

[3] I. Chawla and S. K. Singh, "An automated approach for bug categorization using fuzzy logic," in Proceedings of the 8th India Software Engineering Conference, ser. ISEC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 90–99. [Online]. Available: https://doi.org/10.1145/2723742.2723751

[4] 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, ser. CASCON '08. New York, NY, USA: Association for Computing Machinery, 2008. [Online]. Available: https://doi.org/10.1145/1463788.1463819

[5] H. Fazayeli, S. M. Syed-Mohamad, and N. S. Md Akhir, "Towards auto-labelling issue reports for pull-based software development using text mining approach," Procedia Computer Science, vol. 161, pp. 585–592, 2019, the Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091931871X

[6] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, pp. 121–130.

[7] M. U. Sarwar, S. Zafar, M. W. Mkaouer, G. S. Walia, and M. Z. Malik, "Multi-label classification of commit messages using transfer learning," in 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2020, pp. 37–42.

[8] 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," 2019.

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

[10] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," arXiv preprint arXiv:1607.01759, 2016.

[11] D. Hu, "An introductory survey on attention mechanisms in nlp problems," in Intelligent Systems and Applications, Y. Bi, R. Bhatia, and S. Kapoor, Eds. Cham: Springer International Publishing, 2020, pp. 432–448.

[12] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[14] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," CoRR, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[15] S. Cass, "The top programming languages: Our latest rankings put python on top-again - [careers]," IEEE Spectrum, vol. 57, no. 8, pp. 22–22, 2020.

[16] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," in 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 334–344.

[17] S. Zafar, M. Z. Malik, and G. S. Walia, "Towards standardizing and improving classification of bug-fix commits," in 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2019, pp. 1–6.

[16] https://github.com/GiorgosKarantonis/