Applying Large Language Models API to Issue Classification Problem

Gabriel Aracena, [1] Kyle Luster, [1] Fabio Santos,[1] Igor Steinmacher,[2] Marco A. Gerosa[2]
[1]*Grand Canyon University, USA,* [2]*Northern Arizona University, USA*

GTeixeiraL@my.gcu.edu, kluster1@my.gcu.edu, fabiomarcos.deabre@gcu.edu, igor.steinmacher@nau.edu, marco.gerosa@nau.edu

***Abstract*—-In the realm of software engineering there are issue reports that need labeling so that programmers can prioritize what they want to work on. There are a few approaches to this problem, the most tedious is manually classifying each issue report. This is not scalable. The other approach involves some sort of automated process which can be found in many Open Source Software (OSS) projects. Although these automated processes might work fine, they require a large amount of data for them to be trained. Our intent is to produce an automated approach that is reliable with a small training dataset. We believe that a large language model is the best way to do this.**

***Index Terms*—Issue Report Classification, Large Language Model, Natural Language Processing, Software Engineering, Labelling, Multi-class Classification, Empirical Study**

I. Introduction

OSS communities depend on newcomers, and helping them to onboard is relevant to OSS project's recruitment process [1]. One initial step made by newcomers is to find tasks (e.q. bugs, features, etc) to work with, and the literature advocates as a crucial phase to determine the future of the newcomer within the project [2, 3, 4, 5]. Adding labels to the issues helps new contributors when they are choosing their tasks [6]. However, manually labeling issues in big projects may not be feasible [7].

Since then many researchers proposed approaches to label issue types automatically to help managers prioritize and allocate better available resources. One noted article uses *fastText* [8, 9] to classify issues to *bug*, *feature* or *question*. Colavito et al. [10], employed SETFIT in last NLBSE competition.

Our study tailored to the tool competition leverages the use of OpenAI API to create a fine-tuned model to classify the three available labels with 82% in F1-Score. OpenAI is the innovative company behind the development of ChatGPT the most notorious of the many Large Language Models (LLM) they have built. An LLM refers to a sophisticated neural network model that undergoes training using extensive datasets, including books, code, articles, and websites. This training enables the model to grasp the inherent patterns and relationships within the language for which it was trained. Consequently, the LLM can produce cohesive content, such as grammatically accurate sentences and paragraphs, replicating human language, or syntactically precise code snippets [11]. Although this Artificial Generative Intelligence (AGI) is understandably a blackbox, in a meaning that we don't know exactly what it is doing, it is capable of adapting incredibly when fine-tuned.

RQ1: To what extent can we predict the issue types by using OpenAI's fine-tuning API? To answer RQ1, we employed 5 multi-class classification models by creating a fine-tuning gpt-3.5-turbo base model Open AI fine-tuning API. Fine-tuning is the process of give a specific and niched trainig for a pre-trained model. Fine-tuning through the OpenAI API is a multi step process that involves simulating conversations with the AGI and telling them their specific response expected. We used the title and body of the training data as part of our prompt and the correct label of training data as the expected response from it.

Overall, we found that by pre-processing the issue title and body we can predict the issue types with a macro average of 82.00% precision, 82.00% recall, and 82.00% F-measure. This yield reached the baseline reported in the competition [12].

II. Related Work

Previous research addressed the issue classification problem by employing many different techniques. Kallis et al. [13], employed $J48$ ML algorithm to predict bug, enhancement and questions, and obtained 0.83 for precision, 0.82 for recall, and 0.83 for F-measure. El Zanaty et al. [14] addressed the misclassification problem for inter and intra-project situations classifying issues as bug/non-bug.

Large language models have been used to architect a service-based software [15] with the support of bots. Documentation is also a prominent area where LLMs may be employed [11]. Nathalia et al. [16] evaluated the performance of code automation using LLMs.

III. Method

Figure 1 depicts our method, composed of the clean phase, training phase, predictions phase, and analyze phase.

A. Describing the Data

The data we've received is encapsulated within a single CSV file, encompassing essential details of issue reports characterized by distinct column headings. These columns include:

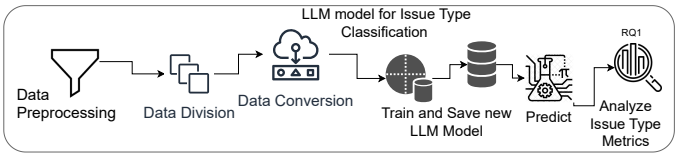

Fig. 1. Method

- repo: This field identifies the specific repository to which the information pertains, with five defined repositories: 'facebook/react', 'tensorflow/tensorflow', 'microsoft/vscode', 'bitcoin/bitcoin', and 'opencv/opencv'.
- created_at: Reflecting the timestamp indicating when a particular issue report was generated.
- label: This categorization distinguishes the nature of the report, labeling it as a bug, feature, or question.
- title: A title for the issue report being created.
- body: A body of text that describes the specifics of the issue report.

The data have five unique repositories with 300 entries each, totaling 1500 entries for the dataset. This data contains issue reports made between January 2022 and September 2023.

B. Noise and Preprocessing

When examining the data for potential noise, we identified several factors contributing to it: extraneous symbols, emojis, autogenerated text, URLs, HTML tags, and excessive whitespace. These elements possess the capacity to skew the sentiment and alter the intended meaning of the text. To mitigate their impact, the majority of these factors were eliminated, while the remaining ones were substituted with a universal identifier carefully chosen to uphold the original message's integrity.

C. Data Division

The data is provided in one, continuous CSV file; however, for the purposes of our project, we segment the data by repository name. This results in five different repositories with 300 issue reports each.

D. Labeled Data

The received data was prelabeled with the categorizations "bug," "feature," and "question," sparing us the need for manual labeling of any data.

E. Converting to a Readable Format

We began our approach by converting each of the repository data frames into JSON line files in preparation for passing them along to the API. First, we created a variable called `user_message`, which begins with a prompt for the model to classify our text, and then the title and body are appended to the prompt. Second, we created an array containing the label 'bug', 'feature', or 'question.' This variable is called `assistant_message`. Third, we created `conversation_message`, which is an object defining the architecture of the text inputted and the model's response to that text. Lastly, this combined text is appended to a JSON line file. These steps are run through iteratively to produce a single JSON line file containing every issue report and its corresponding label, in the format of a conversation.

F. Invoking the API

The API invoked in our code is OpenAI's fine-tuning API, specifically the gpt-3.5-turbo model, used for natural language understanding and generation tasks. This API enables interaction with language models capable of understanding prompts and generating coherent and contextually relevant responses. This API is invoked through OpenAI's Python library.

The interaction with the API occurs through the openai.chat.completions.create() method, where a user prompt is constructed, specifying a task for the model. This involves classifying the issues into categories like 'feature,' 'bug,' or 'question' based on their title and body text, as mentioned in the previous section. The constructed prompt is then sent to the ChatGPT model using the create() method, which returns a generated response or prediction regarding the classification of the pull request. The API invocation involves parameters such as the model to use, the maximum number of tokens to generate, and the temperature for controlling the randomness of responses.

G. Model Fine-tuning

Fine-tuning a model using the OpenAI's API is a multi-step process that requires an understanding of the documentation. As previously mentioned, we created JSON line files for each repository since each repository would need its own fine-tuned model. For each repository, we then uploaded the conversational data to the client server, which is our training file. Then, we created a fine-tuning job, which calls for the base model to be fine-tuned by passing the training file. Every job then enters the queue in the cloud and we retrieve the status of each job every 30 min to see when they are ready. After around 5 hours they were all ready to be used and the server returned us the fine-tuned model ID, for example the facebook/react repo specific model has the ID of *ft:gpt-3.5-turbo-0613:gcucst440:fb-issueclassifier:8LLGMnAI*.

We utilized default parameters when initially fine-tuning each model, which sets the learning rate multiplier to auto, the number of epochs to auto, and batch size to auto. The auto learning rate multiplier will vary depending on the batch size and will also vary depending on the size of the training set which is the conversational data generated. In our case, we calculated the auto batch size to be around 60 and the learning rate to be around 0.1, those two hyper-parameters can be modified but we did not, perhaps that is a possibility of improvement in the future. The number of epochs has a default of 3 epochs, we also used the default by later retraining it for 7 more epochs on each model to improve training and produce better results.

H. Performance Evaluation Metrics

For our performance evaluation metrics, we iterated over the result of each issue on the test dataset and created two lists, one of the true labels and one of the predicted labels of our model. If those lists we were capable of generating 5 confusion matrices, one for each repo/model, and then 1 combined confusion matrix for the overall result. By gathering the values of True Positive (TP), True Negative (TN), False Positive (FN), and False Negative (FN) we were capable of calculating Precision, Recall, and F1-Score of every model and a combined result.

I. Getting the results

For every model, we created a new CSV file with the confusion matrix and with the metrics. After the 5 models were evaluated we then created a combined CSV file with the combined values of TP, TN, FP, and FN.

IV. Results

To answer our RQ1. To what extent can we predict the issue types by using ChatGPT fine-tuning API?

As Table I shows, when we tested different repo datasets to their respective fine-tuned model their results varied from 76.65% to 86.15%. This can be explained by the differences in how issues were written and the particularities of each repositorey. We could observe improvements in terms of precision, recall, and F-measure. When using TITLE and BODY combined we reached overall Precision, Recall, and F1-score of 82.00%

TABLE I
RESULTS PER PROJECT

Repository Metrics Comparison	Precision	Recall	F1
facebook/react	0.8635	0.8600	0.8579
tensorflow/tensorflow	0.8695	0.8600	0.8615
microsoft/vscode	0.8029	0.8000	0.8001
bitcoin/bitcoin	0.7679	0.7700	0.7665
opencv/opencv	0.8134	0.8100	0.8107
overall	0.8200	0.8200	0.8200

RQ1 Summary. It is possible to predict the GitHub Issue labels with precision of 82.27%, recall of 82%, and F-measure of 82.13% using fine-tuned gpt-3.5-turbo base models, with TITLE and BODY as features.

V. Discussion

Observing the complete results (Table II) something really fascinating catches our eyes, why are the metrics so different regarding both the repository and the label? The question type of issue was by far the worst on (almost) every repository. Why is that? To be honest, it seems to us that question is a poorly labeled name for GitHub issues, why is that? It is just a very broad label that consequently causes users to wrongfully label issues as a question and they could be better labeled as something else. When analyzing the data ourselves and checking the question label issues it is very hard even to define a question label issue when reading the title and body. When compared to feature and bug, it is clear that bug issues are not very well defined.

TABLE II
DETAILED ISSUE REPORT CLASSIFICATION COMPLETE TABLE: METRICS BY REPO AND BY LABEL

Repository	Label	Precision	Recall	F1
facebook/react	bug	0.8333	0.95	0.8878
facebook/react	feature	0.8557	0.89	0.8725
facebook/react	question	0.9024	0.74	0.8132
facebook/react	average	0.86	0.86	**0.86**
tensorflow/tensorflow	bug	0.9167	0.88	0.8980
tensorflow/tensorflow	feature	0.9310	0.81	0.8663
tensorflow/tensorflow	question	0.7607	0.89	0.8203
tensorflow/tensorflow	average	0.8695	0.86	**0.8615**
microsoft/vscode	bug	0.8539	0.76	0.8042
microsoft/vscode	feature	0.8	0.84	0.8195
microsoft/vscode	question	0.7547	0.8	0.7767
microsoft/vscode	average	0.8029	0.8	**0.8001**
bitcoin/bitcoin	bug	0.7339	0.8	0.7656
bitcoin/bitcoin	feature	0.8318	0.89	0.8599
bitcoin/bitcoin	question	0.7381	0.62	0.6739
bitcoin/bitcoin	average	0.7679	0.77	**0.7665**
opencv/opencv	bug	0.7455	0.82	0.7810
opencv/opencv	feature	0.8511	0.8	0.8247
opencv/opencv	question	0.8438	0.81	0.8265
opencv/opencv	average	0.8134	0.81	**0.8107**
overall	bug	0.8167	0.842	0.8296
overall	feature	0.8539	0.846	0.8499
overall	question	0.7999	0.772	0.7857
overall	average	0.8227	0.82	**0.8213**

Similarly, when comparing the metrics on different repositories it is clear that the bitcoin repository obtained awful results when compared to the others. When compared to the F1 score of the tensorflow repo that obtained the highest metric (86.15%) the F1 score of the bitcoin repo is almost 10% worse than it (76.65%). Why are the results so different between different repositories? Well, after further analysis we concluded that it is due to bad labeling and description of the issues by the developers that work on each repository. We concluded that by comparing our results with the baseline metrics and saw that the baseline model had the same issue, so it is obvious to us that our Bitcoin model did badly for no reason, and when we analyzed personally the title and body of the issues on the bitcoin repo we saw that a lot of them were just unclear and poorly written.

What are the difficulties in labeling? Depending on the repository and on the developer that is opening an issue there are many concepts that might vary among all of them. As I said above, the definition of each label is one of them, on the GitHub Facebook repository there are 12 different types of possible labels so it might be hard for an unexperienced user to properly label the issue they are opening and sometimes they might even label at one thing but describe as another due to ignorance and inexperience.

Looking at the confusion matrix below in Table III we can also infer the results obtained and understand that indeed our metrics were right. The models have varying degrees of

success in classifying different types of issues (bugs, features, questions) across the different repositories. The model performs consistently in identifying 'bug' labels across repositories, with TP ranging from 80 to 89, suggesting that the features used by the model are good indicators of this class. The models seem to perform well on 'feature' classification, with relatively high TP but also higher FP compared to 'bug' classification. This could indicate confusion between 'feature' and other types of issues, leading to more false alarms. There is a notable variance in the model's ability to correctly classify 'question' labels, with the lowest TP observed in the 'bitcoin/bitcoin' repository.

The 'facebook/react' repository has relatively balanced classification across all labels, indicating that the model may have learned distinctive features of each label well in this context. The 'tensorflow/tensorflow' repository has the highest FP for 'question' classification, suggesting the potential overclassification of this label. The 'microsoft/vscode' repository shows a higher tendency to miss 'feature' and 'question' issues (as indicated by higher FN).

TABLE III
CONFUSION MATRIX FOR ALL PROJECT AND LABELS

Repository	Label	TP	FP	FN	TN
facebook/react	bug	89	15	11	185
facebook/react	feature	95	19	5	181
facebook/react	question	74	8	26	192
tensorflow/tensorflow	bug	81	6	19	194
tensorflow/tensorflow	feature	88	8	12	192
tensorflow/tensorflow	question	89	28	11	172
microsoft/vscode	bug	84	21	16	179
microsoft/vscode	feature	76	13	24	187
microsoft/vscode	question	80	26	20	174
bitcoin/bitcoin	bug	89	18	11	182
bitcoin/bitcoin	feature	80	29	20	171
bitcoin/bitcoin	question	62	22	38	178
opencv/opencv	bug	80	14	20	186
opencv/opencv	feature	82	28	18	172
opencv/opencv	question	81	15	19	185

Our models could likely benefit from additional tuning or training data to improve classification, especially for 'question' labels. Addressing the imbalance between FP and FN across different labels could help improve model performance. This might involve re-evaluating the features used for classification or introducing class weights during training.

VI. CONCLUSION

In conclusion, this study represents a significant step forward in the application of large language models, specifically OpenAI's gpt-3.5-turbo, for the classification of GitHub issue reports. By fine-tuning models on datasets from five distinct repositories, we demonstrated the feasibility and efficiency of this approach. Our methodology, focusing on data preprocessing and model fine-tuning, yielded an average F1-score of 82%, underscoring the model's effectiveness in classifying issues into 'bug', 'feature', or 'question' categories. This performance, however, varied across repositories, revealing the nuanced nature of issue report classification.

One of the key findings was the variability in performance based on the nature of the data in each repository. This underscores the need for tailored approaches when applying language models in different contexts. Furthermore, the study highlighted challenges in classifying 'question' labels due to their often ambiguous nature. This points to a broader issue in the standardization of labeling practices within the GitHub community.

ACKNOWLEDGMENT

This work is partially supported by the NAU computer science research team. We also appreciate Professor Isac Artzi for letting us use the VR Lab facility on our research.

REFERENCES

[1] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information and Software Technology*, vol. 59, pp. 67–85, 2015.

[2] J. Wang and A. Sarma, "Which bug should i fix: helping new developers onboard a new project," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 2011, pp. 76–79.

[3] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1379–1392.

[4] C. Stanik, L. Montgomery, D. Martens, D. Fucci, and W. Maalej, "A simple nlp-based approach to support onboarding and retention in open source communities," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 172–182.

[5] I. Steinmacher, T. U. Conte, and M. A. Gerosa, "Understanding and supporting the choice of an appropriate task to start with in open source software communities," in *2015 48th Hawaii International Conference on System Sciences*. IEEE, 2015, pp. 5299–5308.

[6] I. Steinmacher, C. Treude, and M. A. Gerosa, "Let me in: Guidelines for the successful onboarding of newcomers to open source projects," *IEEE Software*, vol. 36, no. 4, pp. 41–49, 2018.

[7] A. Barcomb, K. Stol, B. Fitzgerald, and D. Riehle, "Managing episodic volunteers in free/libre/open source software communities," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[8] 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 2019, Cleve-*

land, OH, USA, September 29 - October 4, 2019. IEEE, 2019, pp. 406–409.

[9] ——, "Predicting issue types on github," *Science of Computer Programming*, vol. 205, p. 102598, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642320302069

[10] G. Colavito, F. Lanubile, and N. Novielli, "Few-shot learning for issue report classification," in *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, 2023, pp. 16–19.

[11] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Software*, vol. 40, no. 3, pp. 4–8, 2023.

[12] R. Kallis, G. Colavito, A. Al-Kaswan, L. Pascarella, O. Chaparro, and P. Rani, "The nlbse'24 tool competition," in *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*, 2024.

[13] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Predicting issue types on github," *Science of Computer Programming*, vol. 205, p. 102598, 2021.

[14] F. El Zanaty, C. Rezk, S. Lijbrink, W. van Bergen, M. Côté, and S. McIntosh, "Automatic recovery of missing issue type labels," *IEEE Software*, 2020.

[15] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, "Towards human-bot collaborative software architecting with chatgpt," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 279–285.

[16] N. Nathalia, A. Paulo, and C. Donald, "Artificial intelligence vs. software engineers: An empirical study on performance and efficiency using chatgpt," in *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*, 2023, pp. 24–33.