Applying Large Language Models to Issue Classification

Gabriel Aracena
GTeixeiraL@my.gcu.edu
garacena@brandsafway.com
Grand Canyon University
BrandSafway
Phoenix, Arizona, USA

Kyle Luster
kluster1@my.gcu.edu
k1luster@itamco.com
Grand Canyon University
ITAMCO
Phoenix, Arizona, USA

Fabio Santos
fabiomarcos.deabre@gcu.edu
Grand Canyon University
Phoenix, Arizona, USA

Igor Steinmacher
igor.steinmacher@nau.edu
Northern Arizona University
Flagstaff, Arizona, USA

Marco A. Gerosa
marco.gerosa@nau.edu
Northern Arizona University
Flagstaff, Arizona, USA

ABSTRACT

Effective prioritization of issue reports in software engineering helps to optimize resource allocation and information recovery. However, manual issue classification is laborious and lacks scalability. As an alternative, many open source software (OSS) projects employ automated processes for this task, yet this relies on substantial datasets for adequate training. This research investigates an automated approach to issue classification based on Generative Pre-trained Transformers (GPT). By leveraging the capabilities of such models, we aim to develop a robust system for prioritizing issue reports accurately, mitigating the necessity for extensive training data while maintaining reliability. In our research, we have developed a GPT-based approach to label issues accurately with a reduced training dataset. By reducing reliance on massive data requirements and focusing on few-shot fine-tuning, we found a more accessible and efficient solution for issue classification. Our model predicted issue labels in individual projects up to 93.2% in precision, 95% in recall, and 89.3% in F1-score.

KEYWORDS

Issue Report Classification, Large Language Model, Natural Language Processing, Software Engineering, Labeling, Multi-class Classification, Empirical Study

ACM Reference Format:
Gabriel Aracena, Kyle Luster, Fabio Santos, Igor Steinmacher, and Marco A. Gerosa. 2024. Applying Large Language Models to Issue Classification. In *2024 ACM/IEEE International Workshop on NL-based Software Engineering (NLBSE '24), April 20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3643787.3648043

NLBSE '24, April 20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0576-2/24/04…$15.00
https://doi.org/10.1145/3643787.3648043

1 INTRODUCTION

The onboarding of newcomers is important to keep open source software (OSS) projects sustainable [12]. One of the initial steps of the onboarding process in an OSS project is to find an appropriate task (e.g., bugs, features, etc.) to work with [11, 15]. To facilitate task selection, communities add labels to the issues to help new contributors find the most appropriate ones [9, 13]. However, labeling issues in big projects is time-consuming and demands efforts from the (already overloaded) maintainers [1].

Recently, researchers proposed approaches to label issue types automatically to help managers prioritize and allocate available resources. Kallis et al. [4, 5] use *fastText* to classify issues as *bug*, *feature*, or *question*. Still, Colavito et al. [2] employed SETFIT in the 2023 NLBSE competition to predict issue types. Santos et al. [10] predicted skills to solve an issue using API domains as a proxy. Their work was extended to use Social Network Analysis (SNA), improving the predictions [8]. Finally, a tool is available to OSS communities to recommend issues based on skills informed by developers [14].

In this study, we leverage OpenAI's Large Language Model (LLM) API to create a fine-tuned model to classify issues as bugs, features, or questions. An LLM is 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 in a language. Consequently, the LLM can produce coherent content, such as generating natural language and producing syntactically correct code [7], and can produce even better results well when fine-tuned. In this paper, we investigate the following research question (RQs):

RQ: To what extent can we predict issue types using OpenAI's fine-tuning API? To answer RQ, we fine-tuned the gpt-3.5-turbo base model provided in the OpenAI API. Fine-tuning is giving specific and niched training for a pre-trained model. Fine-tuning through the OpenAI API is a multi-step process that involves simulating conversations with the LLM and providing the expected response. We used the title and body of each issue as part of the prompt and the correct label as the expected response.

Overall, we found that pre-processing the issue title and body can predict the issue types with a macro average of 82.8% F1-score. This yield surpassed the baseline reported in the competition [3].

2 METHOD

Figure 1 depicts our method, composed of data preprocessing, model implementation and training, model evaluation, and analysis.

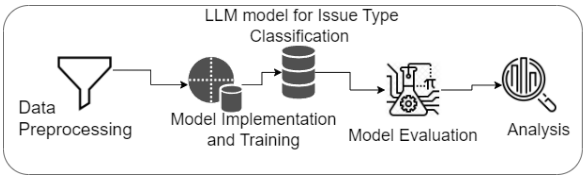

Figure 1: Method

2.1 Data Preprocessing and Cleaning

2.1.1 Introduction to the Dataset. We used the dataset for the NLBSE'24 Tool Competition on Issue Report Classification [3], which is a collection of 3,000 labeled issue reports extracted from five open-source projects. The data was collected over 21 months, from January 2022 to September 2023, covering various issues.

2.1.2 Attributes of Each Issue Report. Four key attributes characterize each issue report in the dataset:

(1) **Repository**: The name of the open-source project from which the issue report was extracted.
(2) **Label**: The category that the issue report falls into.
(3) **Title**: The title of the issue report.
(4) **Body**: The main content of the issue report.

2.1.3 Noise Removal and Data Preparation. **Introduction** Our research utilized two cleaning methods. While the second method aimed to improve upon the first, we observed that some repositories yielded better results with the initial approach. Consequently, we opted to apply the first method to those repositories. Further research could optimize these cleaning methods.

Method 1: Our method involves standardizing text, removing non-essential elements (e.g., double quotes, repository-specific strings, emojis, URLs), and limiting word length to 20 characters. Miller et al. discovered that words exceeding 20 characters make up less than 1% of English words [6]. In our data, we found lengthy alphanumeric strings lacking relevance, leading us to exclude strings surpassing 20 characters to reduce noise.

Method 2: Method 2 mirrors Method 1 but with tailored adjustments. It eliminates new repository-specific strings, transforms URLs, HTML tags, usernames, and images into universal identifiers (e.g., <URL>), and removes unnecessary Markdown syntax, retaining only essential tokens for analysis.

2.1.4 Data Segmentation and Labeling. The dataset was split into five repositories, each with 300 categorized issue reports.

2.1.5 Format Conversion for Model Input. Repository training data frames were converted to JSON line files structured as conversations and sent to the API. Each file includes a prompt (`user_message`): *"Classify, IN ONLY 1 WORD, the following GitHub issue as 'feature', 'bug', or 'question' based on its title and body*:". The second part (`assistant_message`) contains the classification ('bug,' 'feature,' or 'question'). Both halves are concatenated and assigned to (`conversation_message`), which is iteratively appended into a single JSON line file.

2.2 Model Implementation and Training

2.2.1 Invoking the API. Our code utilizes OpenAI's fine-tuning API, specifically the gpt-3.5-turbo model, known for its proficiency in natural language understanding and generation. This API, accessible through OpenAI's Python library, allows interaction with language models, understanding prompts, and generating coherent and relevant responses.

Firstly, we invoke the OpenAI API. Next, the training files are uploaded to the server. Using these files, gpt-3.5-turbo creates fine-tuned models for each repository. To test the model, the interaction with the API occurs through the `create()`[1] method, where a user prompt is constructed. Here, we reuse the prompt from the training cycle, append the testing data, and pass it to the API. The constructed prompt returns its classification of the issue. 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. The model utilized was the fine-tuned model corresponding to its testing dataset, and the maximum number of tokens to be returned was set to 1. The strings 'feature,' 'bug,' and 'label' were all verified by the OpenAI Tokenizer [2] to have 1 token each. The temperature was set to 0.0 to minimize randomness in the answers, providing consistent results.

2.2.2 Model Fine-tuning Process. Fine-tuning a model through OpenAI's API involves a systematic multi-step procedure that requires a comprehensive grasp of the API documentation. As previously outlined, the fine-tuning process was tailored for each repository, demanding dedicated models for enhanced performance.

As explained in the section 2.1.5, we created JSON line files customized for individual repositories. These files were uploaded to OpenAI's server to serve as fine-tuning training data. To start the fine-tuning process, we initiated a job to refine the base model using the specified training file. Each job was queued in the cloud environment. After around 5 hours, all jobs were finished, and the fine-tuned models were prepared for use.

Upon completion, the server assigned unique identifiers for each fine-tuned model. For instance, the model tailored for the 'facebook/react' repository was assigned an ID.

The default hyperparameters were initially used during the fine-tuning process, automatically configuring the learning rate multiplier, number of epochs, and batch size. All models began with the default number of 3 epochs; however, as we conducted our training, step metrics revealed room for improvement in certain models. Given this, each model's epoch count was tailored for optimal performance using grid search. Future improvements might involve fine-tuning other hyperparameters for optimization.

2.3 Performance Evaluation Metrics

In assessing the effectiveness of our fine-tuned models for issue classification, we focused on three key metrics: precision, recall, and F1-score; recommended per competition guidelines [3].

[1] openai.chat.completions.create()
[2] https://platform.openai.com/tokenizer

Precision: It gauges the accuracy of the positive predictions made by the model. It is the ratio of correctly predicted positive observations to the total predicted positives.

Recall: Recall assesses the model's ability to identify all relevant instances within a dataset. It is the ratio of correctly predicted positive observations to all observations.

F1-Score: The F1-score is the harmonic mean of precision and recall.

2.4 Obtaining the results

Our study calculated these metrics for all models by comparing the model's predicted labels against the ground truth from the testing dataset. To ensure reliability and correctness across all models, we utilized the *sklearn.metrics* library to calculate them. The precision, recall, and F1-score collectively offer a holistic view of the model's performance, enabling us to assess its accuracy, robustness, and reliability in classifying issues across different repositories.

We computed metrics for all repositories, then calculated their average by each metric and label, as shown in table 3.

3 RESULTS

To answer our RQ (to what extent can we predict issue types using OpenAI's fine-tuning API?), we tested five fine-tuned models.

Table 1 shows the average F1-score results varied from 76.65% to 87.08%. This can be explained by the differences in how issues were written and the particularities of each repository. Overall, when using TITLE and BODY combined we reached overall precision of 83.24%, recall of 82.87%, and F1-score of 82.80% see table 3

As seen in 2.1.3 two different cleaning methods were used among the models. We initially trained and got the metrics for all projects with Method 1, but after seeing room for improvement, we created and utilized Method 2. However, 2/5 models did not do better with Method 2 so we decided to use Method 1 on them. Further study and analysis would have to be performed for a better evaluation.

RQ Summary. It is possible to predict the issue labels with the precision of 83.24%, recall of 82.9%, and F1 of 82.8% using fine-tuned gpt-3.5-turbo base models, with TITLE and BODY as features. [3]

4 DISCUSSION

Observing the results (Table 1), issues labeled as questions were by far the worst on (almost) every repository. It seems that question is a bad label name for GitHub issues. We suspect questions are very broad labels that consequently cause users to label issues as questions wrongfully when they could be better labeled as something else. When analyzing the data, it is very hard even for humans to classify a question label issue when reading the title and body. One example is a Facebook issue of the testing dataset where even though the issue is labeled as a question, the title is "Bug: useRef can not return a persist ref object". Many other issues are labeled as questions but have "Bug" in the title. Additionally, some issues don't have a question in their description, like this other Facebook issue on the test dataset: "language and translation i'm sure that you translated your react site by Google the worst result at all i hope you correct it...".

[3]Check the full repository at https://github.com/G4BE-334/NLBSE-issue-report-classification

Table 1: Issue Report Classification: Metrics by repository and by label

CM = Cleaning Method ; E = Total Epochs ; P = Precision ; R = Recall

Repo	CM	E	Label	P	R	F1
facebook	1	3	bug	0.8333	**0.9500**	0.8878
facebook	1	3	feature	0.8557	0.8900	0.8725
facebook	1	3	question	0.9024	0.7400	0.8132
facebook	1	3	average	0.8635	0.8600	0.8579
tensorflow	2	10	bug	0.9072	0.8800	**0.8934**
tensorflow	2	10	feature	**0.9318**	0.8200	0.8723
tensorflow	2	10	question	0.7913	0.9100	0.8465
tensorflow	2	10	average	**0.8768**	**0.8700**	**0.8708**
microsoft	1	6	bug	0.8511	0.8000	0.8247
microsoft	1	6	feature	0.8131	0.8700	0.8406
microsoft	1	6	question	0.7980	0.7900	0.7938
microsoft	1	6	average	0.8207	0.8200	0.8198
bitcoin	1	3	bug	0.7339	0.8000	0.7656
bitcoin	1	3	feature	0.8318	0.8900	0.8599
bitcoin	1	3	question	0.7381	0.6200	0.6739
bitcoin	1	3	average	0.7679	0.7700	0.7665
opencv	2	6	bug	0.7288	0.8600	0.7890
opencv	2	6	feature	0.9091	0.8000	0.8511
opencv	2	6	question	0.8617	0.8100	0.8351
opencv	2	6	average	0.8332	0.8233	0.8250

Similarly, when comparing the metrics on different repositories, it is clear that the results for the 'bitcoin' repository were worse when compared to the other repositories. When we compare the F1-score with the 'tensorflow' repository (87.08%), the F1-score of the bitcoin repository is more than 10% worse (76.65%). Why are the results so different across different repositories? After further analysis, we concluded it is due to bad labeling and description of the issues by the developers who 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. Additionally, many of the Bitcoin issues were written without much clarity.

What are the difficulties in labeling? Each repository has specific concepts, technologies, and domain-related topics that vary from one repository to another. Moreover, the issue description style is also different, making automated labeling approaches more challenging.

Looking at the confusion matrix in Table 2, we can see that 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 and higher FP than '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 over-classification of this label. The 'microsoft/vscode' repository tends to miss 'feature' and 'question' issues (as indicated by higher FN).

Table 2: Confusion Matrix for all projects 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	82	6	18	194
tensorflow/tensorflow	feature	88	9	12	191
tensorflow/tensorflow	question	91	24	9	176
microsoft/vscode	bug	87	20	13	180
microsoft/vscode	feature	80	14	20	186
microsoft/vscode	question	79	20	21	180
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	8	20	192
opencv/opencv	feature	86	32	14	168
opencv/opencv	question	81	13	19	187

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 and tuning the learning rate multiplier.

Compared to the baseline results our model performed slightly better overall. The baseline model utilized the *all-mpnet-base-v2* sentence transformer developed by hugging faces as their base model and fine-tuned it with SetFitTrainer. The baseline model, as seen in table 3, had an overall average F1-score of 82.7% when we got 82.8%. The average F1-score on the 'feature' and 'question' labels for the baseline model was 85.58% and 78.27%, respectively. Meanwhile, our models got 85.93% and 79.25% average F1-score, respectively.

5 CONCLUSION

In conclusion, this study applies large language models, specifically OpenAI's gpt-3.5-turbo, to classify 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 82.8%, barely surpassing the baseline 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 problem in the standardization of labeling practices within the GitHub community.

Table 3: Overall metrics comparison table

Model	Metric	bug	feature	question	average
Baseline	P	0.8464	0.8448	0.8001	0.8305
	R	0.8400	0.8700	0.7700	0.8267
	F1	0.8426	0.8558	0.7827	0.8270
Our	P	0.8109	**0.8683**	**0.8183**	**0.8324**
	R	**0.8580**	0.8540	**0.7740**	**0.8287**
	F1	0.8321	**0.8593**	**0.7925**	**0.8280**

ACKNOWLEDGMENTS

Dr. Isac Artzi of GCU partially supported this work. This work was also supported by the National Science Foundation under Grant Numbers 2236198, 2247929, 2303042.

REFERENCES

[1] A. Barcomb, K. Stol, B. Fitzgerald, and D. Riehle. 2020. Managing Episodic Volunteers in Free/Libre/Open Source Software Communities. *IEEE Transactions on Software Engineering* (2020), 1–1.

[2] Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. 2023. Few-Shot Learning for Issue Report Classification. In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*. 16–19. https://doi.org/10.1109/NLBSE59153.2023.00011

[3] Rafael Kallis, Giuseppe Colavito, Ali Al-Kaswan, Luca Pascarella, Oscar Chaparro, and Pooja Rani. 2024. The NLBSE'24 Tool Competition. In *Proceedings of The 3rd International Workshop on Natural Language-based Software Engineering (NLBSE'24)*.

[4] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE, 406–409. https://doi.org/10.1109/ICSME.2019.00070

[5] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2021. Predicting issue types on GitHub. *Science of Computer Programming* 205 (2021), 102598. https://doi.org/10.1016/j.scico.2020.102598

[6] Georges A Miller, Edwin Broomell Newman, and Elizabeth A Friedman. 1958. Length-frequency statistics for written English. *Information and control* 1, 4 (1958), 370–389.

[7] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (2023), 4–8.

[8] Fabio Santos, Jacob Penney, João Felipe Pimentel, Igor Wiese, Bianca Trinkenreich, Igor Steinmacher, and Marco A Gerosa. 2023. Tell Me Who Are You Talking to and I Will Tell You What Issues Need Your Skills. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*.

[9] Fabio Santos, Bianca Trinkenreich, João Felipe Nicolati Pimentel, Igor Wiese, Igor Steinmacher, Anita Sarma, and Marco Gerosa. 2022. How to choose a task? Mismatches in perspectives of newcomers and existing contributors. *Empirical Software Engineering and Measurement* (2022).

[10] Fabio Santos, Igor Wiese, Bianca Trinkenreich, Igor Steinmacher, Anita Sarma, and Marco Gerosa. 2021. Can I Solve It? Identifying APIs Required to Complete OSS Task. (2021).

[11] Igor Steinmacher, Tayana Uchôa Conte, and Marco Aurélio Gerosa. 2015. 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, 5299–5308.

[12] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.

[13] Igor Steinmacher, Christoph Treude, and Marco Aurelio Gerosa. 2018. Let me in: Guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software* 36, 4 (2018), 41–49.

[14] Joseph Vargovich, Fabio Santos, Jacob Penney, Marco Aurélio Gerosa, and Igor Steinmacher. 2023. GiveMeLabeledIssues: An Open Source Issue Recommendation System. In *20th IEEE/ACM International Conference on Mining Software Repositories, MSR 2023, Melbourne, Australia, May 15-16, 2023*. IEEE, 402–406. https://doi.org/10.1109/MSR59073.2023.00061

[15] Jianguo Wang and Anita Sarma. 2011. 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, 76–79.