

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/377303067>

Applying Large Language Models API to Issue Classification Problem

Preprint · January 2024

DOI: 10.13140/RG.2.2.29579.67362

CITATIONS

0

READS

227

5 authors, including:



Gabriel Aracena

Grand Canyon University

1 PUBLICATION 0 CITATIONS

SEE PROFILE



Fabio Marcos de Abreu Santos

GCU

30 PUBLICATIONS 60 CITATIONS

SEE PROFILE



Marco Aurelio Gerosa

Northern Arizona University

281 PUBLICATIONS 5,892 CITATIONS

SEE PROFILE

Applying Large Language Models API to Issue Classification Problem

Gabriel Aracena
GTeixeiraL@my.gcu.edu
Grand Canyon University
Phoenix, Arizona, USA

Kyle Luster
kluster1@my.gcu.edu
Grand Canyon University
Phoenix, Arizona, USA

Fabio Santos
fabiomarcos.deabre@my.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 is crucial in software engineering to optimize resource allocation and address critical problems promptly. However, the manual classification of issue reports for prioritization is laborious and lacks scalability. Alternatively, many open source software (OSS) projects employ automated processes for this task, albeit relying on substantial datasets for adequate training. This research seeks to devise an automated approach that ensures reliability in issue prioritization, even when trained on smaller datasets. Our proposed methodology harnesses the power of Generative Pre-trained Transformers (GPT), recognizing their potential to efficiently handle this task. 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 reliable GPT-based approach to accurately label and prioritize issue reports with a reduced training dataset. By reducing reliance on massive data requirements and focusing on few-shot fine-tuning, our methodology offers a more accessible and efficient solution for issue prioritization in software engineering. Our model predicted issue types 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 API to Issue Classification Problem. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024)*. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

ICSE 2024, April 2024, Lisbon, Portugal

© 2024 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

1 INTRODUCTION

The onboarding of newcomers is important to keep OSS projects sustainable [11]. 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. The literature shows that this is a crucial step to determine the future of the newcomer in the project [10, 14]. One strategy adopted by the communities is to add labels to the issues to help new contributors find the most appropriate ones [8, 12]. However, labeling issues in big projects is time-consuming and demands efforts from the (already overloaded) maintainers [1].

More recently, many researchers proposed approaches to label issue types automatically to help managers prioritize and allocate better 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 last NLBSE competition to predict issue types. Santos et al. [9] predicted skills to solve an issue using API domains as a proxy. The work was extended to use Social Network Analysis (SNA), improving the predictions [7]. Finally, a tool is available to OSS communities to recommend issues based on skills informed by developers [13].

In this study, we leverage OpenAI API to create a fine-tuned model to classify issues as bugs, features, or questions. We reached an F1-score of 82.8%. 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 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 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 [6] and it is capable of adapting incredibly when fine-tuned.

RQ1: To what extent can we predict the issue types using OpenAI's fine-tuning API? To answer RQ1, we fine-tuned the gpt-3.5-turbo base model provided in the OpenAI API. Fine-tuning is the process of 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 telling them the expected response. We used the title and body of the issues in our training data as part of our prompt and the correct label as the expected response.

Overall, we found that by pre-processing the issue title and body, we can predict the issue types with a macro average of 83.24% precision, 82.87% recall, and 82.8% F-measure. This yield barely surpassed the baseline reported in the competition [3].

2 METHOD

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

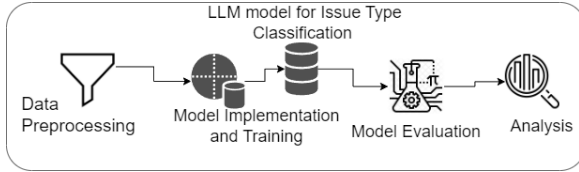


Figure 1: Method

2.1 Data Preprocessing and Cleaning

2.1.1 Introduction to the Dataset. The dataset for the *NLBSE'24 Tool Competition on Issue Report Classification* is a collection of 3,000 labeled issue reports, which have been extracted from five open-source projects. The data was collected over 21 months, from January 2022 to September 2023, ensuring a wide range of issues and scenarios are covered.

2.1.2 Attributes of Each Issue Report. Each issue report in the dataset is characterized by four key attributes:

- (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 Labeling and Exclusions. Each issue has only one label; labeled either bug, feature, or question.

2.1.4 Noise Removal and Data Preparation.

- **Method 1:** The method involves the removal of double quotation marks, elimination of specific string patterns, lowercase conversion for standardization, and removal of emojis, URLs, HTML tags, special characters, and punctuation. The method also addressed consecutive whitespace and restricted word length to 20 characters, as any string longer than that is not likely to be a word but noise.
- **Method 2:** The second method mirrors Method 1's cleaning process with a handful of appropriate modifications. It eliminates more string patterns, modifies handling URLs and HTML tags by substituting them with universal identifiers (<URL> and <HTML_TAG>), and replaces usernames and image links with their own identifiers, <USER> and <IMAGE>, respectively. Lastly, it strips Markdown syntax from the text, striving to preserve only the tokens necessary for analysis.

2.1.5 Data Segmentation and Labeling. The CSV file was split into five repositories, each with 300 categorized issue reports ("bug," "feature," or "question"), and the data provided was pre-labeled.

2.1.6 Format Conversion for Model Input. Repository training data frames were converted into JSON line files, structured as a conversation, and handed off to the API. Each file contains a prompt, (user_message). We decided to use the prompt, "Classify, IN ONLY 1 WORD, the following GitHub issue as 'feature', 'bug', or 'question' based on its title and body:". The second half of the conversation is called (assistant_message), and it contains the model's classification ('bug,' 'feature,' or 'question'). Both halves are concatenated and assigned to the variable (conversation_message), these variables are then appended iteratively into a single JSON line file.

2.2 Model Implementation and Training

2.2.1 Invoking the API. The API invoked in our code is OpenAI's fine-tuning API. Our chosen model is the gpt-3.5-turbo model 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. We invoked the API using OpenAI's Python library.

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 repo. These models will be retrieved for future use. 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—all of which have been tuned to yield optimal results. 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 (<https://platform.openai.com/tokenizer>) 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 "Format Conversion for Model Input" section, we created JSON line files customized for individual repositories. These files were uploaded to OpenAI's server to serve as training data for fine-tuning the models. 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, leading us to regularly check their statuses every 30 minutes. 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

¹`openai.chat.completions.create()`

'facebook/react' repository was assigned the ID *ft:gpt-3.5-turbo-0613:gcucst440:fb-issueclassifier:8LLGMnAI*.

Initially, the default hyperparameters were 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. Future improvements might involve fine-tuning other hyperparameters for optimization.

2.3 Performance Evaluation Metrics

We obtained our results by comparing the predictions with the ground truth. To do this, two lists were created, one with the correct labels from the testing dataset and one with the predicted labels from the fine-tuned model. We used `precision_score`, `recall_score`, and `f1_score` from *sklearn.metrics* to standardize the results obtained across all repositories. Using the metrics functions from *sklearn* and the lists, we were capable of calculating precision, recall, and f1-score for all models.

2.4 Getting the results

After computing metrics for all repositories, we calculated the average for each metric and each label to obtain the overall results present in table 1.

3 RESULTS

To answer our RQ1, (to what extent can we predict the issue types using OpenAI's fine-tuning API?) we tested different repo datasets to their respective fine-tuned model.

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%

RQ1 Summary. It is possible to predict the GitHub Issue labels with precision of 83.24%, recall of 82.87%, and F-measure of 82.8% using fine-tuned gpt-3.5-turbo base models, with TITLE and BODY as features.²

4 DISCUSSION

Observing the complete results (Table 1) something fascinating caught our eyes: why are the metrics so different regarding both the repository and the label? The issues labeled as questions were by far the worst on (almost) every repository. Why is that? It seems that question is a bad label name for GitHub issues. It seems to be related to the fact that question is just a very broad label that consequently causes users to label issues as a question wrongfully, and they could be better labeled as something else. When analyzing the data 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 question issues are not very well defined.

Table 1: Detailed Issue Report Classification Complete Table: Metrics by Repo 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
overall	NA	NA	bug	0.8109	0.8580	0.8321
overall	NA	NA	feature	0.8683	0.8540	0.8593
overall	NA	NA	question	0.8183	0.7740	0.7925
overall	NA	NA	average	0.8324	0.8287	0.8280

Similarly, when comparing the metrics on different repositories it is clear that the results achieved for the bitcoin repository were worse when compared to the other repos. When we compare the F1-score with the tensorflow repo (87.08%) the F1-score of the bitcoin repo is more than 10% worse (76.65%). Why are the results so different across different repositories? After further analysis, we concluded that it is due to bad labeling and description of the issues by the developers who work on each repository. We were able to conclude that by comparing our results with the baseline metrics and saw that the baseline model had the same issue, so it is clear to us that the model performed poorly because of the data. Additionally, we saw that many of the Bitcoin issues were written without much clarity, solidifying our hypothesis.

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

Looking at the confusion matrix below in Table 2, we can see 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

²Check the full repository at <https://github.com/G4BE-334/NLBSE-issue-report-classification>

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 or introducing class weights during training.

When compared to the baseline model results provided by the NLBSE Tool Competition Department our model performed slightly better overall and on some specific repos. 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 had an overall average f1-score of 82.7% when we got 82.8%. The average f1-score on the tensorflow, bitcoin, and opencv repos for the baseline model was 86.44%, 75.55%, and 81.73% respectively. Meanwhile, our models got 87.08%, 76.65%, and 82.5% average f1-score on the same repos respectively.

5 CONCLUSION

In conclusion, this study represents a significant step in applying 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 issue in the standardization of labeling practices within the GitHub community.

ACKNOWLEDGMENTS

This work is supported in part by the NAU computer science research team, as well as Dr. Isac Artzi of GCU.

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] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (2023), 4–8.
- [7] 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)*.
- [8] 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).
- [9] 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).
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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>
- [14] 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.