Improving GitHub Issue Management with NLP-Based Automatic Labeling

Coauthor #1
Department of Computer Science
North Carolina State University
xxx@ncsu.edu

Coathor #2
Department of Computer Science
North Carolina State University
xxx@ncsu.edu

Bennett Zug
Department of Computer Science
North Carolina State University
cbzug@ncsu.edu

1 Problem Statement

In modern software development, bug management tools like GitHub Issues play a central role in coordinating work, tracking bugs, and planning new features. One of the more useful but sometimes underutilized features of GitHub Issues is labeling, which categorizes issues into types such as bug, enhancement, or documentation. Labels are crucial for helping developers and project managers prioritize tasks, delegate responsibilities, and maintain an organized workflow.

However, in many open-source projects, a large number of issues are posted without labels. This often occurs because contributors creating the issue are unfamiliar with the project's labeling scheme, or because project maintainers are too busy to manually review and assign labels. Even when labels are applied, they can be inconsistent, for example, one contributor may describe an issue as a "bug", while maintainers might classify it as an "enhancement". Such subjectivity reduces the reliability of labels as an organizational tool.

At scale, in projects with thousands of open issues, these challenges are amplified, making issue triaging slow and error-prone. The specific problem we address is the lack of consistent and complete labeling of the text within newly posted GitHub issues. Our proposed solution is to apply natural language processing (NLP) to automatically suggest or assign labels directly from the issue's text content. Our hypothesis is that by automatically labeling issue text with NLP, the proportion of unlabeled or inconsistently labeled issues will be significantly reduced compared to relying on manual labeling alone.

An example scenario is a project manager is overseeing an open-source project with over 2000 active issues. They rely heavily on GitHub's labeling system to filter for high-priority bugs before each release cycle. However, nearly half of the issues submitted don't have any labels. The project manager spends hours each week reviewing issues manually to apply correct labels to eventually begin prioritizing work to developers. If an NLP-based auto-labeling system were in place, the project manager could instantly label issues, saving time.

Another example scenario is a developer who is unfamiliar with issue labeling standards for an open-source GitHub repository that they are contributing to. They need to determine an accurate label for their issue that is consistent with other issues on that repository, so that other contributors can easily understand the category of the issue, and efficiently organize all unresolved issues.

2 Importance of the Problem

Accurate labeling of GitHub issues is very useful for managing large repositories, especially in open-source projects with many contributors. Our NLP tool automatically suggests labels based on issue text, reducing manual work, improving triaging speed, and ensuring consistent labels. This helps maintainers focus on development while enabling contributors' issues to be addressed more efficiently.

3 Methodology

We built upon existing research related to the problem of issue classification. Bharadwaj and Kadam (2022) apply BERT-style classifiers specifically to GitHub issue labeling, as part of the NLBSE 2022 competition. Their work and the competition data serve as a reproducible reference: we reused their labeled data, preprocessing, and baseline model setup in our initial work, and compared our fine-tuned BERT variants against published results. The associated competition data serves as a great training and evaluation set, including approximately 803,000 issues sourced from GitHub.

To reproduce their results, we utilized NCSU's Virtual Computing Lab to train the model on an Nvidia L40S. We achieved a similar F1 score of 0.865 after about 2 epochs. Their model uses a RoBERTa architecture, with a custom classification head that integrates both semantic and structural features (Liu et al., 2019). The model predicts three classes: bug, enhancement, and question. To log and track the results of that run, we used Weights & Biases, an AI developer platform. Through this platform, we were able to log metrics like loss, F1 score, and system resource utilization. We also used the platform to save our model weights when the F1 score reaches a new max, to be used later for evaluation and inference.

While RoBERTa offers high performance, it is computationally expensive. To address the need for a more lightweight and deployable solution, we transitioned our architecture to DistilBERT (Sanh et al., 2020). To ensure consistency, we trained this model on the identical resources and hyper-parameters. We achieved an F1 score of 0.858 on this model, which is a negligible degradation of approximately 0.01 compared to the F1 score achieved by the baseline RoBERTa model. We found during training the RoBERTa model that our percentage GPU allocation was 50.87%. While training the DistilBERT model, this dropped to 28.72%. Given the Nvidia L40S's 48gb of memory, this equates to 24.4gb and 13.8gb respectively. This strong reduction in memory makes our model a more viable candidate for deployment in resource-limited environments. Despite this reduction in complexity, we feel that the model performs well with respect to the baseline. With these encouraging results, we decided to use this model as the baseline for our fine-tuning approach.

As per our problem statement, a project manager overseeing an open-source project with many unlabeled issues would benefit greatly from this model. They could deploy the model on accessible hardware and automatically label issues that are unlabeled. We wanted to further explore this idea, and see if we could improve our results by focusing on specific repositories rather than generalizing for every repository. To do this, we considered three options: fine-tuning a pre-trained model like DistilBERT, fine-tuning our existing model that was already fine-tuned on the competition dataset, or performing parameter-efficient fine-tuning on that existing model. The issue with the first approach is that it only sees our small, specific dataset and isn't able to learn general issue labeling patterns from the large competition dataset. The issue with the second approach is that by continuing to update all of the weights, the model may exhibit "catastrophic forgetting", where we'd expect to see a significant degradation in performance on the original data it was trained on. Thus, we decided to implement parameter-efficient fine-tuning on our DistilBERT model that was trained on the competition dataset. The technique we used is called "Low-rank adaptation" (LoRA), where we add new trainable parameters to most layers in the network, and freeze existing weights (Hu et al., 2021). This allows the model to learn new data, without significantly degrading in accuracy with respect to the original data it was trained on. This fine-tuning approach was also very performant, training in around ten minutes.

To fine-tune our model, we needed a new dataset that only focuses on a specific domain. We selected the following GitHub repositories that all use TypeScript: "Redocly/redoc", "APIs-guru/graphql-voyager", "palantir/plottable", "ionic-team/ionic-framework", "excaliburjs/Excalibur", "stenciljs/core", "langfuse/langfuse", "cyclejs/cyclejs", "themesberg/flowbite", "digitallyinduced/thin-

backend", "dnote/dnote", "n8n-io/n8n", "tridactyl/tridactyl", "treehousedev/treehouse", and "NativeScript/NativeScript". We randomly selected 10,000 issues to train on and 500 issues to evaluate our model. We began by evaluating our DistilBERT model on this data as a baseline, which achieved an F1 score of 0.862. After fine-tuning that model on the training data, we achieved an F1 score of 0.896. Thus, we feel that we have shown that we have been able to take our widely-adapted model and fine-tune it for a specific domain, improving performance by 3.5%. To return to our example of a project manager overseeing an open-source project, we feel that our results are a good indicator that our model could be fine-tuned to improve labeling accuracy for their specific data. We propose that future work could include fine-tuning a similar model for one specific repository. If that repository has sufficient data, then we'd expect further improvements in performance.

To demo and test our working DistilBERT model, we also developed a simple python command line script. As previously mentioned, we save the weights for the model with the highest F1 score to the Weights & Biases platform, so it can be conveniently downloaded for inference. Our script loads that model and allows the user to enter an issue title and body, and then it outputs the predicted label and the probabilities of each label. We propose that future work could include utilizing GitHub's API to retrieve unlabeled issues, predict an appropriate label, and then either allow the user to review those predicted labels or automatically apply them.

4 Alternative Approaches and Rationale

The first alternative we explored was manually labeling issues. Each team member labeled 150 issues that were randomly selected from GH Archive and originally posted between October 28 and October 30 2025, which ensures that they are not issues found within the training set. We also reused this same set of 150 issues to test our other methods, allowing us to compare their effectiveness against a consistent baseline.Ultimately, human-manual labeling proved to be the most ineffective method as it was not only incredibly inefficient from a time and human-resource perspective but also produced the lowest performance of all three approaches we tested, with an accuracy of 0.8810 and an F1 score of 0.8879. This result was surprising, as we initially predicted there would be a tradeoff in which manual labeling would yield higher accuracy at the cost of time. Instead, this approach lost on both fronts, making it unrealistic and impractical for real-world use.

The second, more realistic alternative we explored was using OpenAI's gpt-5-mini to classify issues directly through prompt engineering. This approach gave us an F1 score of 0.92 on the same 150 issues. While this LLM-based approach initially seemed promising due to its generalization capabilities and ability to interpret ambiguous issue descriptions, it proved inconsistent in practice: small variations in wording often produced wildly different labels, and relying on external API calls introduced latency, cost, and scalability limitations. Its only major advantage was that it could process non-English issues with passable accuracy, whereas both our model and human evaluators struggled with issue text written in other languages.

By contrast, our DistilBERT-based classifier produced more stable predictions with the highest F1 score of 0.9365 on those 150 issues. This method presented to be the most accurate and cost-effective solution to our problem compared to the alternatives. Throughout this process, we encountered several challenges that shaped our decision, including unexpected label imbalance (specifically the rarity of the "question" label) which significantly impacted minority-class recall, as well as surprising variability in human evaluations that highlighted the subjective nature of issue categorization. These challenges reinforced the value of a tunable, locally deployable and scalable model, ultimately justifying our choice to rely on a fine-tuned DistilBERT classifier rather than an LLM-based or manually labeled approach.

5 Evaluation

To evaluate our approach, we used a combination of quantitative and qualitative methods designed to measure both the predictive performance of the model and its practical usefulness in real-world software development workflows. Quantitatively, we assessed the classifier using standard metrics including accuracy, precision, recall, and F1 score across all issue types.

Our first use case involved establishing a baseline model trained on a broad, diverse dataset of over 800k issues collected from a wide range of GitHub repositories. This baseline served as a

control for all subsequent evaluations. By measuring performance on this general-purpose model, we were able to determine how well a single classifier could generalize across issue types and labeling conventions. The results from this baseline informed our expectations for how the model might behave when applied to new or unfamiliar repositories. Our live demo will showcase this use case, the baseline model, where we input an issue title and body and it shows the predicted label along with probabilities of each label.

Our second use case involved direct comparison against 150 human-labeled issues. After team members independently labeled 50 previously unseen and randomly selected issues, we compared these annotations with both the model's predictions and the original ground-truth labels from the repositories. Alongside this, we also had an LLM label these same 150 issues. This setup allowed us to evaluate not only the model's performance but also the inherent difficulty of the classification task itself. As already mentioned in the previous section, we found our model to not only be faster but also more accurate than humans and LLMs at labeling issues.

Our third use case focused on training, testing and fine-tuning the model within a more domain-specific environment. We selected a group of 15 TypeScript-based demo repositories' issues and used them to evaluate how the model performs when tailored to a narrower ecosystem. This case allowed us to explore a scenario where a development team trains the classifier specifically on its own repositories to capture project-specific language patterns, coding terminology, and labeling conventions. By comparing the performance of this specialized model to the baseline, we assessed the benefits and trade-offs of domain-specific training. This helped us understand whether fine-tuning on similar repositories leads to measurable improvements or whether the general-purpose model remains sufficient for most workflows. Alongside this we also utilized an LLM to label these issues to compare results.

6 Results and Findings

Model	**NLBSE 2021**			**Typescript repos test set**			**Manual Labeling**		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
RoBERTa	**0.865**	**0.865**	**0.865**	–	–	–	–	–	–
DistilBERT	0.858	0.858	0.858	0.862	0.862	0.862	**0.9365**	**0.9365**	**0.9365**
DistilBERT + LoRA	–	–	–	**0.896**	**0.896**	**0.896**	0.9206	0.9206	0.9206
gpt-5-mini	–	–	–	0.8469	0.8540	0.8470	0.9321	0.9206	0.9238
Human Labeling	–	–	–	–	–	–	0.8872	0.8810	0.8740

Table 1: Performance comparison across datasets and models.

We found that small language models can be effectively used to label issues in a GitHub repository. Our baseline approach of finetuning RoBERTa achieved an F1 of 0.865 on the NLBSE competition dataset. We then applied the same fine-tuning procedure to DistillBERT, and observed only very slight performance degradation, while using roughly half of the compute resources. This DistillBERT model was then evaluated against several other models on two test sets: a 500 issue test set of issues from various TypeScript repositories, and a manually labeled dataset composed of 150 issues from random projects.

On the TypeScript test set, our full DistillBERT finetune achieved an F1 score of 0.862. We then performed a LoRA adaptation step using data from the same repositories as the test set. This achieved an F1 score of 0.896, about a 3% improvement. These two models were then evaluated on the dataset of manual labels, and they performed about equally, demonstrating that the gains from the LoRA adaptation were domain specific instead of a general increase in performance.

We also compared these models to an LLM prompt based classifier, as well as human labeling in the case of the manual labeling dataset. On the TypeScript test set, the LLM produced an F1 of 0.847, below both versions of DistilBERT. On the manually labeled dataset, the LLM reached an F1 of 0.9238, slightly below the DistilBERT baseline. Human annotators achieved an F1 of 0.874 when evaluated against the original GitHub labels. In both cases, the small language models performed equal to or better than the LLM and human labels, while requiring far less computational resources.

Overall, our results show that small language models can be competitive with the performance of larger LLMs on small domain specific software engineering tasks, while having distinct advantages in cost and speed.

7 Potential Impact and Applications

After completing our project, we now have a clearer understanding of how our automated issue-labeling system could influence real-world software development workflows and the people who rely on them. The primary stakeholders who would be affected include open-source maintainers, contributors, project managers, and organizations that use GitHub for coordination and project tracking. For maintainers, our results show that a model-based labeling tool can significantly reduce the amount of time spent manually reviewing and categorizing new issues, especially in large repositories where issue volume can easily become overwhelming. Contributors also stand to benefit: automated, consistent labeling makes their reports more visible and reduces confusion for newcomers who may not be familiar with a project's existing triage labeling conventions. Organizations, particularly those managing multiple software teams, could use such a system to improve workflow consistency, streamline triage processes, and reduce bottlenecks in backlog management.

However, evaluating our system also revealed several potential harms and risks. Incorrect or biased labels could deprioritize important issues, especially those written by less experienced contributors or those who use non-standard English, potentially discouraging participation and reinforcing inequities within open-source communities. Mislabeling accessibility issues, security concerns, or urgent bugs could delay fixes and harm end users who depend on timely responses. Moreover, while our model is designed specifically for GitHub Issues, in principle it could be misused to categorize or filter user-generated text in other contexts, which opens the possibility of discriminatory outcomes. For example, those writing issues with non-standard English due to their background may be at a higher risk of issues mislabeling than those with standard'English. Suddenly a high priority bug becomes a low priority question and, if deployed without transparency or oversight, this model can contain subtle discrimination.

To address harms arising from intended use, we suggest that the most effective mitigation strategy is ensuring that automated labels remain recommendations rather than authoritative decisions. Providing maintainers with confidence scores, opportunities to override labels, and documentation of known limitations helps prevent over-reliance on the model. In addition, regular audits and retraining with diverse datasets can also reduce systemic biases. To mitigate potential misuse, we can limit the scope of the tool by clearly documenting that it is intended only for GitHub-style issue classification and by discouraging applications to other forms of communication while also adding rate limits to reduce the risk of harmful deployments. Through these measures, we can help ensure that the system enhances collaboration and efficiency without unintentionally harming contributors or enabling misuse.

References

Shikhar Bharadwaj and Tushar Kadam. 2022. Github issue classification using BERT-style models. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering (NLBSE'22)*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.