Recommending Good First Issues in GitHub OSS Projects

Wenxin Xiao*
School of Computer Science, Peking University, and Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
wenxin.xiao@stu.pku.edu.cn

Hao He*
School of Computer Science, Peking University, and Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
heh@pku.edu.cn

Weiwei Xu
School of Computer Science and Technology, Soochow University
Suzhou, China
wwxu99@stu.suda.edu.cn

Xin Tan
School of Computer Science, Peking University, and Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
tanxin16@pku.edu.cn

Jinhao Dong
School of Computer Science, Peking University, and Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
dongjinhao@stu.pku.edu.cn

Minghui Zhou†
School of Computer Science, Peking University, and Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing, China
zhmh@pku.edu.cn

ABSTRACT

Attracting and retaining newcomers is vital for the sustainability of an open-source software project. However, it is difficult for newcomers to locate suitable development tasks, while existing "Good First Issues" (GFI) in GitHub are often insufficient and inappropriate. In this paper, we propose RecGFI, an effective practical approach for the recommendation of good first issues to newcomers, which can be used to relieve maintainers' burden and help newcomers onboard. RecGFI models an issue with features from multiple dimensions (content, background, and dynamics) and uses an XGBoost classifier to generate its probability of being a GFI. To evaluate RecGFI, we collect 53,510 resolved issues among 100 GitHub projects and carefully restore their historical states to build ground truth datasets. Our evaluation shows that RecGFI can achieve up to 0.853 AUC in the ground truth dataset and outperforms alternative models. Our interpretable analysis of the trained model further reveals interesting observations about GFI characteristics. Finally, we report latest issues (without GFI-signaling labels but recommended as GFI by our approach) to project maintainers among which 16 are confirmed as real GFIs and five have been resolved by a newcomer.

CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**; **Maintaining software**.

KEYWORDS

open-source software, onboarding, good first issues

*Both authors contributed equally in this work
†Corresponding Author

ICSE 2022, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN xxx.
https://doi.org/10.1145/3510003.3510196

ACM Reference Format:
Wenxin Xiao, Hao He, Weiwei Xu, Xin Tan, Jinhao Dong, and Minghui Zhou. 2022. Recommending Good First Issues in GitHub OSS Projects. In *The 44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510196

1 INTRODUCTION

Open Source Software (OSS) has become the infrastructure of our society. One possible explanation for the success of OSS is its unique development model [37, 40], in which the source code is open for everyone and everyone can contribute back to the source code. Although such development model permits the emergence of enormous OSS serving a wide spectrum of requirements, it also poses challenges on the sustainability of OSS. Existing studies have shown that OSS are prone to sustainability failures [9, 58], which may impact a large number of downstream clients and cause severe losses.

One major challenge of OSS sustainability is to attract and retain capable newcomers [65–67]. Successful onboarding into OSS projects can be very difficult, especially for developers who are new to open-source and for projects with a long development history [32]. To help newcomers become familiar with an OSS project, GitHub provides a list of best practices, such as providing READMEs, formulating contribution guidelines, etc [18]. Even if sufficient documentation is provided, it is still very challenging for newcomers to locate suitable development tasks to start with [49]. For example, not needed pull requests are among the most common cause for rejected code in OSS projects because newcomers often submit "superseded/duplicated pull-requests" [51]. Recently, GitHub recommends project maintainers to label issues as "Good First Issues" (GFIs) [14], which is an explicit signal showing that this issue is suitable and welcome for newcomers to solve.[1]

However, several recent studies have raised alarming concerns about the current GFI mechanism [1, 23, 56]. First, manually labeled GFIs are often highly insufficient [1, 56], which indicates that many

[1]We use the term Good First Issue (GFI) and "issues suitable for newcomers" interchangeably throughout this paper.

actual GFIs do not have a GFI label and may not be discovered by newcomers. Second, the cognitive mismatch between newcomers and veterans hinders the effectiveness of GFIs [56], which indicates that labeling GFIs can be a challenging task and existing labeled GFIs may be inappropriate for newcomers. To tackle the insufficiency and inappropriateness of current GFIs, we envision the use of an automated approach for GFI recommendation. Such an approach can not only ease the burden of maintainers from labeling GFIs, but can also provide the most suitable task for newcomers so that they are more likely to succeed in making their initial contributions.

In this paper, we propose RecGFI, a machine learning approach toward this vision. RecGFI learns from historical issues *actually resolved* by newcomers instead of learning from issues having a GFI label, as these labels are scarce and potentially inappropriate [1, 56]. For each historical issue, RecGFI extracts features from heterogeneous sources of information including issue content, background, and dynamics. RecGFI further uses an XGBoost classifier [8] to learn the difference between issues resolved by newcomers and non-newcomers. Then, for all open issues, RecGFI predicts their probability of being GFIs using the trained model and returns the most probable GFIs for human inspection. Newcomers can browse through the recommendations to find suitable tasks, while project maintainers can inspect and confirm the recommendations (e.g., by adding labels) with reduced manual effort.

To evaluate RecGFI, we collect 53,510 issues from 100 GitHub projects using the GHTorrent dataset (dump 2021-03-06) [19] and GitHub REST API [15]. To simulate realistic scenarios and avoid leakage of future data [57], we create two ground truth datasets by carefully restoring features from two time points: the time when the issue is created and the time before the issue is being worked on. Our performance evaluation shows that RecGFI can achieve up to 0.801 AUC at 1st time point, and up to 0.853 AUC at 2nd time point. RecGFI also outperforms other baseline approaches and alternatives with less features. We further interpret the trained model using Local Interpretable Model-agnostic Explanations (LIME) [41] and analyze statistics on the dataset to study characteristics of issues suitable for newcomers. Our analysis reveals that RecGFI is more likely to predict an issue as a GFI if 1) the reporter has "just enough" experience in commits and more experience in reporting issues; 2) experienced developers participate in the issue (e.g., by adding labels, etc); and 3) the project has an active owner and more recently onboarded newcomers. Finally, to evaluate the practical usefulness of RecGFI, we collect 511 latest issues from the 100 GitHub projects, report 60 issues (without GFI-signaling labels) to project maintainers, and monitor how these issues are resolved. At the time of writing, we receive response for 33 issues and 16 are confirmed as real GFIs. After the 16 issues are labeled as GFIs by project maintainers, three have already been resolved by a newcomer. Even for the 17 issues denied by project maintainers, two have also been resolved by a newcomer. We share a replication package at https://zenodo.org/record/5881117#.YeliUEBBwlI.

2 THE GFI RECOMMENDATION PROBLEM

2.1 Problem in Reality

It is well-known that making the first contribution to an OSS project is hard. For example, a newcomer recently complains that:

There's a lack of resources out there for someone who wants to get started contributing to open-source mostly because they find it hard to make the first contribution and write the first piece of code.[2]

One major reason hindering newcomers' contribution to open-source is the lack of sufficient labeled GFIs in most GitHub OSS projects. In a sample of 105 highly popular GitHub projects, only 46 projects have GFIs and only 1.5% of all issues in the 46 projects are labeled as GFIs [1]; another study reports a median proportion of only 4% for projects with labeled GFIs [56]. The insufficiency of GFIs is already preventing newcomers from participating in OSS projects. For example, a newcomer complains on Reddit that:

Looking to contribute to open source, but issues labeled good-first-issue are all taken...I've had a hard time finding issues that are good for beginners that aren't completely taken over already.[3]

We also observe similar requests in GitHub issues because the newcomer cannot find any GFIs:

I had gone through the whole project and I'm really excited to contribute to it if is there any good first issues then please assign me.[4]

The lack of labeled GFIs in an OSS project may harm its sustainability in the long term. However, a typical OSS project generally have hundreds to thousands of open issues and maintainers may lack time to manually evaluate whether these issues are GFIs. For example, `netlify/netlify-cms` has 591 open issues at the time of writing but only 97 are labeled as GFIs. Even if `netlify/netlify-cms` already have a much higher GFI portion compared with the median 4% [56], some real GFIs may still be missed by maintainers and can not be discovered by newcomers. For example, the issue in Figure 1 is actually suitable for newcomers but nobody has added any GFI-signaling label.[5] In addition, the manual labeling of GFIs may be even challenging for project maintainers due to the cognitive mismatch between newcomers and veterans. A well-understood construct of the "zone of proximal development" [59] describes the case where experts are usually not effective at training or teaching novices. This observation is supported by data: 40.9% of GFIs are not solved by newcomers and 31.2% of newcomers fail to solve a GFI even after several attempts [56].

In such scenarios, an automated recommendation approach will be useful: newcomers can browse through the recommendations to find issues to work on, while project maintainers can check the recommendations and label them with much less effort. In fact, the issue in Figure 1 is only added with a GFI label after it is discovered by our RecGFI approach (see Section 4.5).

2.2 Problem Formulation

Given a list of open issues in a GitHub project, the goal of GFI recommendation is to find a subset of issues that are likely suitable for newcomers. Therefore, we formulate the problem to be resolved as binary classification. The objective is to utilize information of historical issues to learn a model $f(\cdot)$ and predict whether an given

[2] https://www.reddit.com/r/opensource/comments/otuhdv/when_you_find_an_opensource_project_you_like_what/
[3] https://www.reddit.com/r/learnprogramming/comments/j48nkn/looking_to_contribute_to_open_source_but_issues/
[4] https://github.com/accordproject/cicero/issues/693
[5] The issue can be accessed at https://github.com/netlify/netlify-cms/issues/5735. In Figure 1, some information (e.g., screenshots) is omitted due to space constraints. The issue is suitable for newcomers because it is relatively simple, has reproduction steps, detailed screenshots, and clear expectation of a resolution [56].

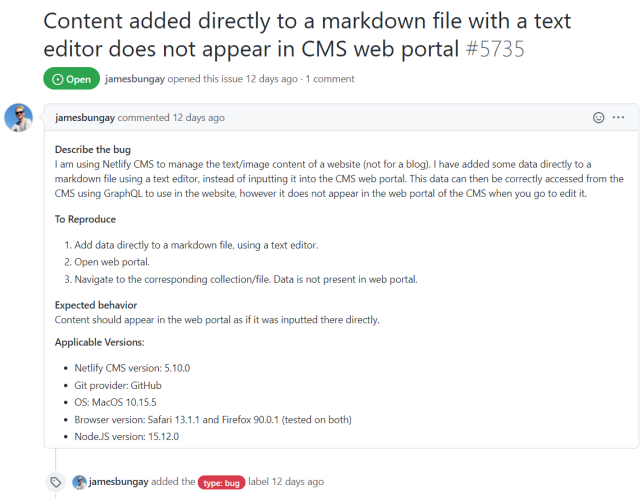


Figure 1: A good first issue without any signaling label.

unresolved issue $\boldsymbol{x}$ is a GFI. The learned model $f(\cdot)$ by this objective can be used in flexible ways. For example, it can be used to recommend a list of issues to newcomers in project websites or suggest project maintainers to inspect label certain issues if the model predicts them to be GFIs.

To learn $f(\cdot)$, we need to define ground truth labels for historical issues. As mentioned in Section 2.1, manually labeled GFIs are often insufficient and inappropriate, and thus not suitable as ground truth labels. In this paper, we propose to learn $f(\cdot)$ to predict *whether an issue will be resolved by a newcomer*. In the historical issues (i.e., the training dataset), we define $f(\boldsymbol{x}) = 1$ if issue $\boldsymbol{x}$ is resolved by a newcomer and $f(\boldsymbol{x}) = 0$ if issue $\boldsymbol{x}$ is not resolved by a newcomer. We define developers as **newcomers** in a project if they has contributed no more than k commits in this project, and we define the developers as **non-newcomers** if otherwise.

We choose to introduce a threshold parameter k for two reasons. First, previous research has shown that not only developers who have not contributed to the project, but also developers who have successfully made contributions seek to resolve GFIs because they believe that their capabilities are insufficient for other issues [56]. Second, the criteria of being a "newcomer" may be different across projects and we want to retain some flexibility in problem formulation. For example, successfully contributing one commit in complex software systems like the Linux Kernel may indicate that the developer is already qualified and experienced enough, while in some other projects making small contributions is much easier. The effect of threshold k on performance will be discussed in Section 4. In real application scenarios, we expect project maintainers to choose a proper k and use the corresponding model for their project. We will provide suggestions on choosing k in Section 5.

3 APPROACH

In this section, we present RecGFI, our solution to the GFI recommendation problem, including how it extracts features from three heterogeneous sources of information and how it uses an XGBoost classifier to learn from these features for GFI classification.

3.1 Feature Engineering

Identification and modeling of viable features is vital for the performance of any recommender systems [45]. RecGFI extracts three dimensions of features from an issue to abstract and learn evidence indicating possible GFIs: Content, Background, and Dynamics.

3.1.1 Content. Intuitively, one can infer whether an issue is a GFI after reading its content, including issue title, description, and labels. Many kinds of information in issue content may be helpful in such inference, such as whether the issue is clear, what kind of issue it is, etc. Therefore, we derive the following features:

Title and Description. Intuitively, issues with longer, clearer, and more detailed description should help newcomers understand and resolve correctly. Therefore, we extract the following statistics from issue title and description: length (i.e., number of words) of issue title (`len_title`), length of issue description (`len_body`), number of URLs (`#urls`), number of code snippets (`#code_snips`), and number of images (`#imgs`). These statistical features are frequently used in previous work to measure the quality and granularity of issue reports [24, 56]. We also compute the following readability metrics for title and description: Coleman-Liau formula (`coleman`), Kincaid Grade Level (`kincaid`), Flesch Reading Ease (`flesch`), and Automated Readability Index (ARI) (`ari`). These readability metrics are also used in previous work to measure the readability of bug reports [7, 12, 34]. Furthermore, we use the training set to find the 50 most frequent keywords from descriptions of issues resolved by newcomers and non-newcomers respectively. We remove common words and get two keyword lists. For description of an issue, we separately count number of words (`#gfi_words` and `#nongfi_words`) in the two keyword lists. Finally, for issue title and description, we compute TF-IDF [44] for the top 50 words with highest term frequency (excluding stop words), forming two 50-dimension feature vectors (`tfidf_title` and `tfidf_description`).

Labels. Labels are an effective approach to attach categorical information to issues, such as topic, priority and development task [54]. Except for generic labels (e.g., bug), a variety of custom labels are widely used in projects [26]. Certain types of labels may indicate an issue as likely suitable for newcomers. To capture evidence from labels, we divide common labels into 12 categories including 1) **Task Type**: Bug, Documentation, Test, Build, Enhancement, Coding, or New Feature; 2) **Difficulty/Priority**: GFI-Signaling, Medium Difficulty, or Difficult/Important; and 3) **Status**: Triaged or Untriaged. To derive label categories and heuristics for automatic categorization of labels, we count the occurrence of all labels in our dataset (Section 4.2), manually categorize frequently-occurring labels, and discuss detection rules until reaching a consensus. For each category, we derive detection rules based on keyword matching. For example, a label is considered as Bug if it contains keyword "bug", or GFI-Signaling if it contains any of the common keywords provided by Tan et al. for newcomers [56], or Untriaged if it contains keywords like "untriaged", "needs triage", etc. For each issue, we count the numbers of labels belonging to the 12 categories and the total number of labels (`#labels`), resulting in a 13-dimensional feature vector.

3.1.2 Background. Apart from issue content, some issue background information may also be useful when inferring whether an

issue is a GFI. In RecGFI, we extract features from the following sources of background information:

Reporter. Developers who have reported newcomer-resolved issues may be more experienced in reporting issues and report better GFIs. On the other hand, veterans may report complex issues or issues missing information necessary for newcomers to resolve this issue. Therefore, we collect the following features to characterize reporter experience in 1) project development: being a newcomer him/herself (`rptr_is_new`), number of commits (`#cmt_proj`), issues (`#issues_proj`), and pull requests (`#pr_proj`) in the project; 2) OSS development in general: number of commits (`#cmt_all`), issues (`#issues_all`), and pull requests (`#pr_all`) in all over the GitHub, number of repositories owned (`#repo`), number of stars received in their own repositories (`#stars_rptr`) and number of followers (`#followers`); and 3) GFI reporting: number of reported issues resolved by non-newcomers divided by number of reported issues resolved by newcomers (`nongfi/gfi`). Additionally, we check if reporters have comments for the issues (`has_comment`) or participated in any kind of events (`has_event`) to measure their participation in their reported issues.

Project. OSS participants tend to seek projects that match task difficulty with their skill level [31], while projects may vary in difficulty, skill requirements, and attractiveness to external developers. Although project background information does not help when recommending within a specific project, they may help calibrate other features and transfer knowledge from other projects in cross-project recommendation scenarios. Therefore, we extract the following project-related features 1) basic information: number of stars (`#stars`), contributors (`#contributors`), commits (`#cmt`) and closed pull requests (`#pr`); 2) owner information: same set of features previously used to characterize reporter experience, excluding `rptr_is_new`; 3) issue-related information: number of open issues (`#iss_open`), median issue close time (`iss_cls_t`), and number/ratio of issues resolved by newcomers (`#gfi_proj` and `:gfi_proj`).

3.1.3 Dynamics. An issue is not static after its creation. It may attract attention from different stakeholders (maintainers, users, etc), be supplemented with more information, or undergo status changes. These dynamics may affect whether an issue is suitable for newcomers and should also be considered during recommendation. Therefore, we extract the following features:

Comments. After issue creation, other developers may comment on issues to add supplementary information, provide guidance/management (project maintainers), or express interest (potential resolvers). For each issue, we count the number of comments (`#comments`) as the first feature. Then, similar to issue title and description, we generate a 50-dimension feature vector based on the TF-IDF computed from comment text (`tfidf_comment`).

Events. The events of an issue reflects change of issue status after its creation. Each event carries different meanings and may be related to all kinds of activities within this GitHub repository.[6] For example, a "subscribed" event indicates the issue is drawing attention from more developers, while a "mentioned" event indicates the issue is complicated and related with other issues. Similar to labels, we first find the top-25 events with most occurrences in the training set. Then, for each issue, we count total number of events (`#events`) and the number of events for each frequently occurring event type, forming a 26-dimension feature vector.

Participants. Similar to the issue reporter, the expertise of issue participants may also play a role in determining whether the issue is a GFI, since they may improve the issue (by adding labels, editing titles, etc) and provide help to newcomers. Therefore, for all labelers and event operators, we compute the same features for characterizing project development expertise and OSS development expertise as the issue reporter. For GFI reporting expertise, we compute the number of reported issues resolved by newcomers (`#gfi_labeler` and `#gfi_evtr`) and ratio of reported issues resolved by newcomers (`:gfi_labeler` and `:gfi_evtr`) . We take the average of all labelers' expertise and the average of all event operators' expertise as two input feature vectors. We also compute the number/ratio of newcomers among event operators (`#event_new` and `:event_new`), and the number/ratio of newcomers among labelers (`#labler_new` and `:labeler_new`) as additional issue participant features.

3.2 The XGBoost Classifier

RecGFI uses the eXtreme Gradient Boosting (XGBoost) [8] classifier to learn from historical ground truth data and predict whether an issue is a GFI. XGBoost is a gradient boosted regression tree approach with several clever optimizations which enables its excellent performance and high scalability. XGBoost has achieved state-of-the-art performance in a diverse range of application scenarios [8] including software engineering tasks (e.g., [61, 63]). XGBoost is also easily interpretable using the generated feature weights and supports automated feature selection during ensemble of trees. We will show in Section 4 that XGBoost is also highly competitive for the GFI recommendation problem, outperforming other machine learning approaches in our ground truth dataset.

Since the number of positive and negative samples is highly imbalanced in the ground truth data (Table 1), RecGFI resamples the training set before training the XGBoost classifier, which is a common processing strategy for dealing with imbalanced data [4, 47]. Specifically, issues belonging to the minor class (i.e., issues resolved by newcomers) are oversampled and issues belonging to the major class (i.e., issues resolved by non-newcomers) are undersampled, so that the proportion of two classes in training set is balanced. Finally, given an unseen issue in the test set (Section 4.3) or collected from GitHub (Section 4.5), RecGFI uses the trained XGBoost classifier to output its probability of being an GFI.

4 EVALUATION

4.1 Research Questions

To test the ability of RecGFI in identifying suitable issues for newcomers and to enrich our understanding on the GFI recommendation problem, we ask the following research questions:

RQ1: How does RecGFI perform in predicting whether an issue is suitable for newcomers? As discussed in Section 2.2, there is no clear boundary of experience to divide newcomers and non-newcomers, so we set a threshold k in newcomer definition. Therefore, we need to evaluate the performance of RecGFI under different k, in both time points, and in different settings. We also present performance of RecGFI and baseline approaches to validate

[6] https://docs.github.com/en/developers/webhooks-and-events/events/issue-event-types

that RecGFI is more competitive, and investigate the contribution of each feature group by comparing RecGFI with its variants.

RQ2: What kind of issues are suitable for newcomers? We take the following two steps to facilitate the understanding of issues suitable for newcomers: 1) use Local Interpretable Model-agnostic Explanations (LIME) [41] to reveal contributions of features in the trained model; 2) statistically analyze the differences between issues resolved by newcomers and other issues.

RQ3: Is RecGFI helpful in a real world setting? Finally, to demonstrate the practical usefulness of our approach, we collect latest open issues and use our model to generate recommended GFIs. Using the recommendations, we conduct a preliminary user study among project maintainers and monitor how these recommended GFIs are actually resolved.

4.2 Data Collection

We choose to build a dataset for the GFI recommendation problem because we are not aware of any prior work that provides a similar dataset. To retrieve engineered open source software projects willing to attract newcomers' attention, we follow the instructions introduced by Munaiah et al. [38] to select representative GitHub repositories. According to the Octoverse Report,[7] JavaScript, Python, Java, TypeScript, C#, PHP, C++ and C are the top eight popular programming languages on GitHub in 2020. We use the latest Libraries.io[8] dataset to filter out the repositories whose primary programming language is one of them and keep 25,793 repositories with more than 100 stars, 5 contributors, 5 forks and 10 open issues. We further use the GHTorrent dataset [19] (dump 2021-03-06) to collect all issues for these projects and locate issues with GFI-signaling labels according to the label set provided by Tan et al. [56]. To locate projects with a strong interest in attracting newcomers, we sort projects by the number of issues with GFI-signaling labels and only keep the top 100 projects. These projects have 1,417,497 issues in total and only 27,445 (1.93%) issues with GFI-signaling labels (this GFI ratio is in line with previous studies [1, 56]).

Since the GHTorrent dataset [19] does not contain sufficient data for our approach (e.g., lacking issue text), we further use the GitHub REST API [15] to collect title, description, labels, comments, and events for all issues and pull requests. Issue events are needed for restoring historical issue state before certain time points and pull requests are used to trace issues to their resolvers. This collection process takes about one month using 15 GitHub API tokens.

Issues on GitHub have different nature such as asking questions, reporting bugs, and proposing features [28]. Not all issues correspond to a specific development task to be assigned to a specific developer. Therefore, we need to locate issues resolved by a specific developer. However, recovering links between issues and the version control system is still an open problem [43], so we resort to a conservative but reliable approach: finding issues closed by a commit or pull request (PR) according to GitHub conventions. In GitHub, developers can link commits/PRs to issues using certain text patterns (e.g., `closes #num`) which also tells GitHub to automatically close the issue if the commit appears in the main branch or the PR is merged [13, 17]. If an issue is closed by a commit, we find the commit using the corresponding commit SHA recorded by GitHub in the issue close event [16]. If an issue is closed by a PR, we scan patterns in PR text (as defined in [17]) to find whether a PR closed this issue. Finally, we are able to collect 53,510 issues and their corresponding resolver in the 100 projects.

When computing features, we need to take time into consideration to avoid leakage of future data which frequently leads to overly optimistic results [57]. To simulate real world scenarios and diversify our dataset, we restore issue historical state at two time points: the time when the issue is created and before some developer starts to work on this issue. RecGFI should work well for both time points since any issue in between the two time points may be a possible GFI for newcomers. We compute the second time point as follows: if an issue is referenced by a pull request or assigned to a specific developer, it indicates that a developer may already be working on it, so we use the first reference or assignment event as the second time point; if the issue has never been referenced or assigned, we set the second time point to the time before the issue is closed. We compute all features at the two time points for learning and prediction. We then compute the number of commits made by the issue resolver before the issue is closed (excluding commits that resolved this issue). This number can be used to distinguish between newcomers/non-newcomers and generate ground truth labels when training RecGFI with a given threshold.

4.3 RQ1: Performance

4.3.1 Performance Metrics. Since RecGFI outputs a probability, the dataset is highly imbalanced (Table 1), and relative ranking of issues matters more than a specific classification threshold, we choose to use **AUC** (Area under the ROC Curve) [20] as the main performance metric. AUC provides an aggregated measure of performance across all classification thresholds and represents the probability that a randomly chosen positive sample will be ranked higher than a randomly chosen negative sample [35]. Additionally, we list accuracy ($\frac{TP+TN}{TP+TN+FP+FN}$) and recall ($\frac{TP}{TP+FN}$) at classification threshold 0.5 as a reference, in which accuracy measures to what extent can RecGFI correctly classify an issue and recall measures to what extent can RecGFI discover all GFIs in the ground truth dataset. We favor recall over precision because precision represents the proportion of issues that are resolved by newcomers among issues recommended by the model. However, issues resolved by non-newcomers may still be suitable for newcomers due to non-newcomers' early discovery and preemption (Section 4.4). Therefore, even an ideal model that perfectly identifies all newcomer-friendly issues may not reach a high precision, and precision may not be an appropriate measure for our dataset. For all experiments (except for Table 2), we use 10-fold cross-validation [53] to measure model performance and record average value for each performance metric.

4.3.2 Performance Under Different Settings. As mentioned in Section 2.2, the "perfect" definition for newcomer may vary across different projects, so RecGFI should perform well for a range of k thresholds. Table 1 shows the AUC, accuracy and recall of RecGFI under different thresholds at two time points. We can observe from Table 1 that the performance of RecGFI slightly increases as k goes larger, but all metrics are generally stable over a range of threshold

[7] https://octoverse.github.com/
[8] https://libraries.io/data

Table 1: Performance under different k.

k	1st time point AUC	Acc	R	2nd time point AUC	Acc	R	# of issues resolved by newcomer/non-newcomer
0	0.792	**0.787**	0.623	0.845	**0.832**	0.674	5,448/48,062
1	0.791	0.773	0.645	0.844	0.817	0.701	6,693/46,817
2	0.795	0.765	0.653	0.846	0.813	0.703	7,480/46,030
3	0.799	0.761	0.680	0.852	0.816	**0.723**	8,247/45,263
4	**0.801**	0.761	**0.690**	**0.853**	0.813	0.721	8,702/44,808

Table 2: Performance when RecGFI is trained with historical issues and validated on latest issues.

k	1st time point AUC	Acc	R	2nd time point AUC	Acc	R
0	0.780	**0.781**	0.596	0.815	**0.813**	0.621
1	0.779	0.759	0.632	0.815	0.794	0.661
2	0.780	0.757	0.630	0.814	0.790	0.668
3	0.786	0.755	0.660	**0.823**	0.790	0.685
4	**0.790**	0.754	**0.674**	0.823	0.788	**0.689**

Table 3: Cross-project performance under different k.

k	1st time point AUC	Acc	R	2nd time point AUC	Acc	R
0	**0.687**	**0.733**	0.485	**0.789**	**0.796**	0.598
1	0.676	0.711	0.506	0.780	0.765	0.627
2	0.689	0.699	**0.569**	0.777	0.756	0.653
3	0.672	0.676	0.561	0.782	0.757	0.660
4	0.687	0.680	0.565	0.785	0.754	**0.669**

values and different time points. To validate that RecGFI can work well in a realistic setting, we sort issues in our dataset by time and divide them into 90% of past issues and 10% of latest issues. Then, we use the former as the training set to train RecGFI and the latter as the test set to evaluate performance. This setting is close to real deployment when RecGFI is trained from historical data and used to recommend GFIs among latest open issues. We summarize the performance results in Table 2, in which only a slight AUC decrease (1.4%∼3.8%) can be observed, indicating that RecGFI should also perform well in real development settings. To investigate whether RecGFI performs well in a cross-project setting, we further show the results of 10-fold cross-validation by project (i.e., issues from 90 projects for training and issues from 10 projects for validation) in Table 3. We observe a higher AUC decrease (6.6%∼15.9%), which indicates that it may be more challenging to transfer learned GFI knowledge to unseen OSS projects.

4.3.3 Performance Comparison with Alternatives. To comprehensively analyze the performance of RecGFI and show the contributions of each data source, we compare with five baseline approaches and several RecGFI variants:

We include the following baseline approaches:

- **Random:** Randomly predicts whether an issue is suitable for newcomers, whose theoretical AUC is 0.5.
- **Stanik et al. [48]:** Extracts features from issue titles and descriptions (lengths, TF-IDFs, and sentiment scores) and uses a random forest model for prediction.
- **Logistic Regression:** Trains a logistic regression model from all RecGFI features using the Python `scikit-learn` package [39].
- **Random Forest:** Trains a random forest model from all RecGFI features using the Python `scikit-learn` package [39].

Table 4: Performance comparisons with baselines. ($k = 0$)

Approaches	1st time point AUC	Acc	R	2nd time point AUC	Acc	R
Random	0.502	0.514	0.489	0.496	0.491	0.504
Stanik et al. [48]	0.604	0.808	0.275	0.604	0.808	0.275
Logistic	0.727	0.620	0.704	0.740	0.614	0.739
Random Forest	0.775	0.846	0.455	0.809	0.864	0.476
Deep Neural Network	0.735	0.643	0.714	0.747	0.682	0.678
RecGFI-Title&Description	0.619	0.708	0.430	0.619	0.708	0.430
RecGFI-Labels	0.505	0.361	0.676	0.694	0.782	0.332
RecGFI-Reporter	0.717	0.670	0.647	0.770	0.716	0.659
RecGFI-Project	0.756	0.745	0.612	0.761	0.756	0.619
RecGFI-Comment	-	-	-	0.582	0.745	0.349
RecGFI-Event	-	-	-	0.590	0.630	0.488
RecGFI-Participant	-	-	-	0.660	0.785	0.403
RecGFI-Abl-Title&Description	0.790	0.774	0.640	0.847	0.830	0.681
RecGFI-Abl-Labels	0.792	0.788	0.617	0.833	0.824	0.657
RecGFI-Abl-Reporter	0.758	0.770	0.583	0.796	0.798	0.611
RecGFI-Abl-Project	0.739	0.739	0.579	0.817	0.807	0.641
RecGFI-Abl-Comment	-	-	-	0.844	0.829	0.674
RecGFI-Abl-Event	-	-	-	0.840	0.827	0.669
RecGFI-Abl-Participant	-	-	-	0.843	0.830	0.681
RecGFI	0.792	0.787	0.623	0.845	0.832	0.674

- **Deep Neural Network:** Uses two bidirectional LSTMs [22] to jointly learn two embeddings for issue title and description, and concatenates the embeddings with remaining numerical features into fully connected layers for prediction.

We include the latter three as baselines because of their effectiveness in previous software engineering work [3, 10, 42].

We also include the following variants of RecGFI:

- **Variants of RecGFI with Single Data Source:** The features used in RecGFI can be summarized into seven data sources, including issue title and description, labels, reporter, project, comments, events, and participants. We keep only features from one of the data sources in turn as input of our model, resulting in seven variants of RecGFI: RecGFI-Title&Description, RecGFI-Labels, RecGFI-Reporter, RecGFI-Project, RecGFI-Comment, RecGFI-Event and RecGFI-Participant.
- **Variants of RecGFI with Ablated Data Source:** For ablation study, we remove each data source from RecGFI in turn, and get the other seven variants of RecGFI, namely RecGFI-Abl-Title&Description, RecGFI-Abl-Labels, RecGFI-Abl-Reporter, RecGFI-Abl-Project, RecGFI-Abl-Comment, RecGFI-Abl-Event, and RecGFI-Abl-Participant.

Because the choice of k depends on the specific needs of projects, we do not preset a "perfect" threshold for the following experiments. We present the performance results with other approaches and RecGFI-variants in Table 4 only for $k = 0$ due to space constraints. As shown in Table 4, RecGFI outperforms other approaches in terms of AUC, indicating that RecGFI has better discrimination ability.[9]

Contrary to our intuition, reporter-related and project-related features alone can achieve higher performance than features from other single data sources. Experience of reporters may be closely related to difficulty of issues they reported and it will be discussed in detail in Section 4.4. For project-related features, possible explanations are: 1) historical newcomer onboarding is a strong indicator of whether an issue will attract a newcomer, and 2) project information can supplement information related to issue difficulty. On the

[9]Note that even if some baselines can achieve higher accuracy or recall at classification threshold 0.5, it does not mean that they are consistently better at all thresholds. The latter property is measured by AUC which we use as the main performance indicator.

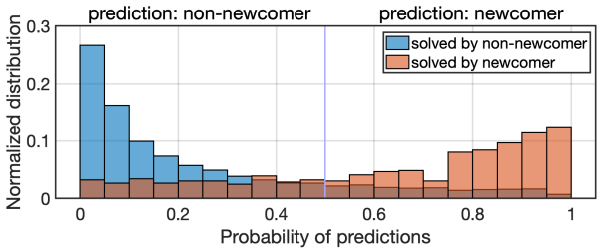


Figure 2: Distribution of predicted probability for issues resolved by non-newcomers and newcomers.

other hand, the ablation study shows that removing features from a single data source (including reporter-related and project-related features) will not cause significant performance decrease in model performance. The reason may be that different data sources contain significant information overlap at the second time point.

At the first time point, RecGFI with only label-related features performs the worst among all variant models, while its performance improves at the second time point. The reason is that 89.68% issues have no labels when they are reported but only 33.01% issues have no labels at the second time point. As time goes by, more information from labels is available and inaccurate labels are corrected.

Figure 2 shows the probability distribution of each prediction for issues resolved by non-newcomers and newcomers with threshold $k = 0$ at the second time point. We can observe that the model successfully learns to distinguish issues suitable/inappropriate for newcomers, though without a clear boundary. This is expected because the notion of "newcomer suitable issue" is inherently ambiguous. If an issue's predicted probability is close to 0 or 1, it is much more likely to be a correct classification.

Summary for RQ1:

RecGFI successfully learns to identify issues resolved by newcomers and outperforms baseline approaches with up to 0.801 AUC (1st time point) and 0.853 AUC (2nd time point).

4.4 RQ2: Outline of Issues Suitable for Newcomers

We use LIME [41] to explain the prediction results of RecGFI with relative contributions of features for single sample. Given a test sample, LIME generates adjacent data around the sample and fits an interpretable linear model locally to learn weights of features. Then, we draw an outline of issues suitable for newcomers through interpretable predictions and statistical analysis of the dataset.

4.4.1 Interpretable Prediction for Feature Analysis. To understand the role of different features of RecGFI in predicting issues within a project, we apply LIME to the prediction results of issues from the Microsoft/vscode[10] project. Figure 3 shows weight distribution of features other than those related to project, label and event types on all test issues of vscode. Outliers are removed from Figure 3. On the whole, the experience level of reporters is negatively correlated to the probability that the reported issue is suitable for newcomers.

[10] https://github.com/Microsoft/vscode

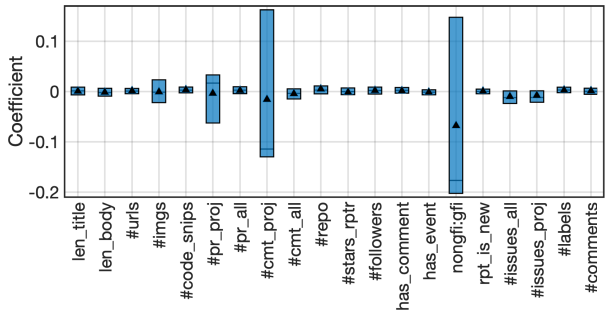


Figure 3: Coefficients of partial features from vscode project given by LIME.

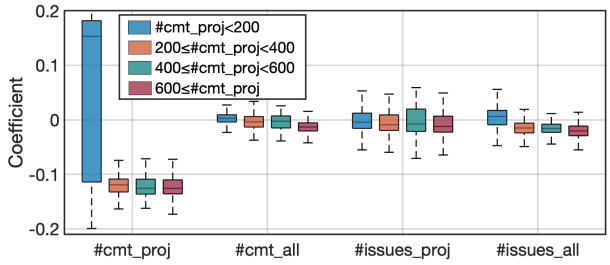


Figure 4: Coefficients of reporter-related features for four groups divided by reporters' commits number in project. The boundaries are 200, 400, and 600 respectively.

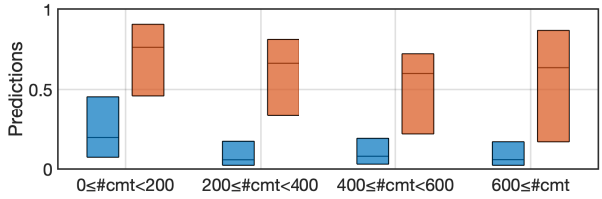


Figure 5: Predicted probability of issues resolved by non-newcomers (blue) and newcomers (orange) under different reporters' commits number in project.

Specifically, developers who have reported more issues and contributed more commits are less likely to report GFIs. Higher ratio of historical non-GFIs also plays a similar role.

To further observe issues reported by developers with different experiences, we divide reporters into four groups according to the number of commits they contributed to the project. The coefficients of reporter-related features are presented in Figure 4. Although in general experience of reporters lowers the probability of prediction, when they have fewer (less than 200) commits in project, more experience means that issues reported by them are more likely to be GFIs. Figure 5 confirms that issues reported by developers with less than 200 commits in project are more likely to be GFIs. In general, "just enough" experience in project leads to a higher probability for the developer to report a GFI.

Intuitively, project-related features are not helpful when predicting issues within the same project at the same time, because these

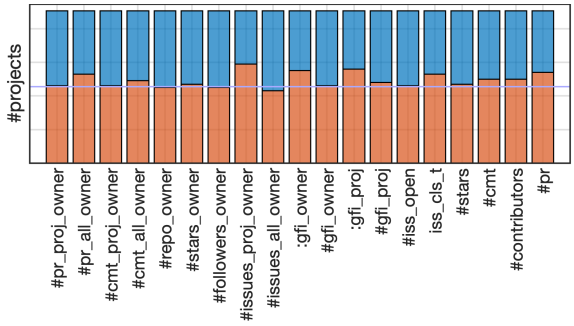


Figure 6: Number of projects with positive (orange) and negative (blue) coefficients for project-related features.

features are the same for the issues. These features are used to adjust the predicted probabilities of issues across projects or different periods of the same project, so that issues can be classified using the same probability threshold. In this way, project maintainers do not need to spend extra effort to adjust probability threshold for different period based on expert knowledge. To show what kind of projects RecGFI tends to provide higher prediction probability, we count the number of projects with positive/negative coefficients for project-related features in Figure 6. Although the trend is not obvious for most features, we can still see that issues from projects with higher ratio of historical GFIs (`:gfi_proj`) and more contributions from project owner (e.g., `issues_proj_owner` and `:gfi_owner`) are more likely to be suitable for newcomers. The reason may be that project owner's investment in project implies that newcomers may get more support to resolve issues. Longer time needed to resolve issues (`iss_cls_t`) indicates that newcomers have more opportunities to complete issues. This inference is consistent with the finding in previous study that GFIs take longer time to be resolved [56].

4.4.2 Comparison of Issues Resolved by Non-Newcomers and Newcomers. We divide all issues in our dataset into two groups according to whether they are resolved by non-newcomers or newcomers. We first test all numerical features (106 in total) of the two groups using Mann–Whitney U test to see if their median values are significantly different [33]. Table 5 shows statistics of partial features with p-value < 0.00047 (~0.05/106, with Bonferroni correction [11]), which indicates that medians of these features in the two groups are significantly different. We can observe that issues resolved by newcomers tend to contain longer descriptions, more hyperlinks, and more code snippets. The statistics of reporters in Table 5 are consistent with the features of reporters discussed in Section 4.4.1. Besides, reporters are more involved in issue-related activities for issues resolved by newcomers. At the project level, projects owned by more experienced developers or companies may have more issues resolved by newcomers. Projects with fewer stars, commits, and more contributors tend to have more GFIs. In other words, active projects in earlier development period may have more GFIs.

For issues resolved by newcomers, developers who label the issues or change the status of the issues have richer development experience. Therefore, we can make a reasonable inference that the participation of veterans is helpful for newcomers to resolve issues. We classify common labels into 12 categories in Section 3.1. Following conclusions can be drawn from Table 6: 1) Documentation tasks are more likely to be suitable for newcomers than Coding tasks; 2) Medium Difficulty tasks are also likely to be suitable for newcomers; 3) even if issues are tagged with GFI-Signaling labels, only 21.23% of them are resolved by newcomers.[11]

Then, we compare the distribution of words in issue descriptions from the two groups. To highlight difference while avoiding project specific terms, we first filter out common stopwords and words occurred in less than 5 projects. After filtering, we further plot two

Table 5: Statistics of features from issues resolved by non-newcomers and newcomers.

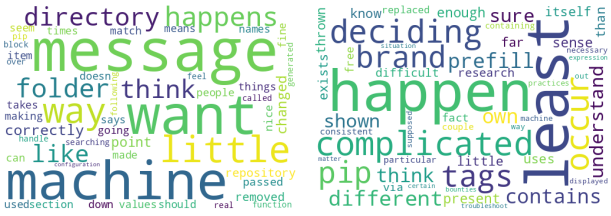
Features	Mean		Median	
	non-newcomer	newcomer	non-newcomer	newcomer
Title and Description				
len_body*	85.82	99.66	59	72
#urls*	1.04	1.13	1	1
#code_snips*	0.58	0.74	0	0
Labels				
#labels*	1.54	1.78	1	2
Reporter				
#pr_proj*	80.65	42.12	8	1
#pr_all*	212.67	134.72	61	19
#cmt_proj*	712.02	242.48	16	0
#cmt_all*	2,994.18	1,595.98	889	286
#issues_proj*	13.91	4.85	1	0
#issues_all*	460.45	235.71	122	34
nongfi/gfi*	60.45	6.42	6.97	0
#repo*	44.10	39.05	23	20
#stars_rptr*	357.82	608.76	7	4
#followers*	211.46	195.12	29	11
has_comment*	0.27	0.33	0	0
has_event*	0.28	0.32	0	0
rptr_is_new*	0.39	0.68	0	1
Project				
#pr_proj_owner*	0.24	0.31	0	0
#pr_all_owner*	3.71	8.13	0	0
#cmt_proj_owner*	3.47	5.71	0	0
#cmt_all_owner*	136.21	311.40	0	0
#issues_proj_owner*	0.54	0.67	0	0
#issues_all_owner*	12.06	30.24	0	0
#repo_owner*	684.44	396.66	83	25
#stars_owner*	123,275.99	100,428.33	17,388	10,991
#iss_open*	7,145.98	6,546.67	4,317	3,971
#gfi_proj*	8.44	15.83	3	6
:gfi_proj*	0.08	0.24	0.02	0.20
#contributors*	395.53	450.05	229	265
#cmt*	18,187.59	14,307.51	12,237	8,210
Comments				
#comments*	2.42	3.02	1	1
Events				
#event_labeled*	4.96	5.56	2	3
#event_subscribed*	1.57	1.80	1	2
#event_referenced*	1.51	1.39	0	0
Participants				
#pr_proj_labeler*	103.87	113.72	10	25
#pr_all_labeler*	400.84	578.57	61	93
#cmt_all_labeler*	4,148.89	4,119.72	688	910
#repo_labeler*	33.41	37.09	9	17
#stars_labeler*	398.87	914.05	1	3
#followers_labeler*	235.70	346.46	12	20
#gfi_labeler*	0.02	0.03	0	0
:gfi_labeler*	0.003	0.01	0	0
#labler_new*	0.18	0.26	0	0
:labler_new*	0.07	0.10	0	0
#pr_proj_evtr*	12.02	13.43	2.92	3.58
#pr_all_evtr*	35.93	49.40	8.41	10.77
#cmt_all_evtr*	417.38	450.05	112	124
#repo_evtr*	3.94	4.48	1.46	2.08
#stars_evtr*	59.81	137.72	0.65	1.15
#followers_evtr*	27.01	40.46	3.40	4.31
#event_new*	0.18	0.26	0	0
:event_new*	0.002	0.004	0	0

* p-value<0.00047.

[11] This ratio is lower than the 40.9% reported by Tan et al. [56] because they use $k = 2$ as newcomer definition while we use $k = 0$ in Table 6.

Table 6: Number and proportion of issues with different labels resolved by non-newcomers and newcomers.

Labels	#issues	resolver: non-newcomer	resolver: newcomer
GFI-Signaling	6,482	5,106 (78.77%)	1,376 (21.23%)
Untriaged	680	544 (80.00%)	136 (20.00%)
Documentation	2,022	1,623 (80.27%)	399 (19.73%)
Medium Difficulty	1,056	856 (81.06%)	200 (18.94%)
All Issues	53,510	48,062 (89.82%)	5,448 (10.18%)
New Feature	2,195	1,974 (89.93%)	221 (10.07%)
Triaged	1,355	1,220 (90.04%)	135 (9.96%)
Test	1,551	1,398 (90.14%)	153 (9.86%)
Enhancement	3,186	2,878 (90.33%)	308 (9.67%)
Bug	13,809	12,489 (90.44%)	1,320 (9.56%)
Build	628	571 (90.92%)	57 (9.08%)
Coding	587	560 (95.40%)	27 (4.60%)
Difficult/Important	1,446	1,402 (96.96%)	44 (3.04%)



(a) Issues resolved by newcomers (b) Issues resolved by non-newcomers

Figure 7: The most frequent words in description of issues resolved by newcomers and non-newcomers.

word clouds for remaining words using the following weighted frequency formulas:

$$weight_{\text{GFI}}(word) = \frac{freq^2_{\text{GFI}}(word)}{freq_{\text{GFI}}(word) + freq_{\text{non-GFI}}(word)}$$

$$weight_{\text{non-GFI}}(word) = \frac{freq^2_{\text{non-GFI}}(word)}{freq_{\text{GFI}}(word) + freq_{\text{non-GFI}}(word)}$$

Here $freq_{\text{GFI}}(word)$ and $freq_{\text{non-GFI}}(word)$ means the frequency of *word* in issues resolved by newcomers and non-newcomers, respectively; and $weight_{\text{GFI}}(word)$ and $weight_{\text{non-GFI}}(word)$ means the weight value assigned to *word* when plotting word clouds for issues resolved by newcomers and non-newcomers, respectively. The two word clouds are shown in Figure 7. For issues resolved by newcomers, words related to small amount of work and fine-grained details tend to be more common, while words related to high difficulty and complex situations tend to be more common in issues resolved by non-newcomers.

4.4.3 Observations of Quickly Resolved Issues. The reasons for issues to be resolved quickly may be that they are easy or they are resolved by veterans. We hypothesize that issues resolved in two hours are easy and not complicated, thus they are very likely to be suitable for newcomers. We choose a strict "short time" (two hours) to ensure that issues are resolved quickly because of simplicity, rather than veterans completing difficult tasks. Nevertheless, it needs to be clear that some issues that are resolved within two hours may not be suitable for newcomers. Table 7 provides statistics of all issues and issues resolved within two hours (hereinafter referred to as easy issues). As can be seen from Table 7, for issues reported by newcomers, there is almost no difference in the proportion of issues resolved by newcomers in easy issues and all issues. For issues reported by non-newcomers, the proportion of easy issues resolved by newcomers (1.15%) is lower than the overall ratio (5.49%), since newcomers do not have enough time to find easy issues. The other reason is that developers, especially those who are experienced in projects, tend to resolve easy issues reported by themselves compared with overall situation (33.64% to 61.05%). While 61.05% of issues reported by non-newcomers are resolved by themselves, the ratio is only 33.64% for issues reported by newcomers. The reason may be that some newcomers are just project users and do not plan to participate in development of project. In addition, newcomers need longer time to resolve issues. Even if they want to resolve issues reported by themselves, they may be preempted by non-newcomers.

The above observations show a dilemma for newcomers. They usually need more time to resolve an issue, but experienced developers have advantages because they can discover issues earlier and resolve issues faster. This actually increases competition and difficulty for newcomers to resolve GFIs. Therefore, it is important to label GFIs early and correctly so that newcomers can locate suitable issues more quickly and more accurately.

Table 7: Statistics of resolvers for all issues in the dataset and issues resolved within two hours, respectively.

Reporter \ Resolver	non-newcomer	newcomer	reporter	#issues
(All issues)				
Newcomer	20,005 (83.97%)	3,818 (16.03%)	2,085 (8.75%)	23,823
Non-newcomer	28,057 (94.51%)	1,630 (5.49%)	9,988 (33.64%)	29,687
(Issues resolved in 2h)				
Newcomer	539 (83.31%)	108 (16.69%)	79 (12.21%)	647
Non-newcomer	1,713 (98.85%)	20 (1.15%)	1,058 (61.05%)	1,733

Summary for RQ2:

RecGFI predicts an issue as more likely to be a GFI if: 1) it is reported by developers with "just enough" experience; 2) it contains participation from experienced developers; and 3) the project has an active owner and more onboarded newcomers.

4.5 RQ3: Real World Evaluation

To demonstrate that RecGFI works on real-world unseen data and also useful under realistic scenarios, we run a script on August 23, 2021 to collect and compute features for open issues with recent activities in the last seven days in the 100 projects. We get 3,130 issues after this step. Then, we use our trained RecGFI model to predict whether these issues are possible GFIs, and we retain 511 issues with prediction probability higher than 0.7. We choose the probability threshold of 0.7 because we do not want to bother project maintainers with too many recommended issues and these issues with higher possibilities are more likely to be real GFIs as demonstrated in Figure 2. Project maintainers can choose an appropriate probability threshold according to the needs of their project. A lower probability threshold means more potential false positives and fewer GFIs omissions, while a higher probability threshold ensure that recommended issues are more likely to be GFIs and more GFIs may be missed.

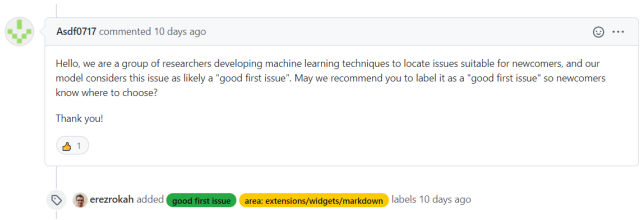


Figure 8: The issue in Section 2.1 is labeled as a GFI by a project maintainer after our recommendation.

Our first goal is to evaluate whether the recommended GFIs are really perceived as GFIs by project maintainers. To achieve this goal, we carefully check the 511 recommended issues by browsing through issue titles, description, labels, comments, and timeline events. We also check project contribution documentation to understand their issue workflow where necessary. As expected, a large number of open issues are not in a suitable state for recommendation: 70 issues still need further information or discussion to be confirmed as a real issue; 115 issues are already being solved by a specific developer; 25 issues do not correspond to a specific development task (i.e., they may be discussions, proposals, etc); finally, 9 issues are closed between script execution and manual inspection. Fortunately, we observe that most projects have custom labels to specify issue states (e.g., "Needs Triage", "In Discussion", "In Progress", etc) and better recommendation can be facilitated by customizing RecGFI to work on a subset of open issues (e.g., by specifying filtering rules on issue labels).

In the remaining 216 issues, 67 (31.0%) already have GFI-signaling labels while 28 (13.0%) have some evidence indicating they are not suitable for newcomers (e.g., difficult, complex, etc). For the remaining 121 issues, we group them by project and report them to project maintainers in the following way: if a project has more than three recommended issues, we open a new issue/discussion in the project to report up to 10 issues as possible GFIs; if the project contribution guideline does not allow opening such issues, we directly comment on up to three issues to suggest adding GFI-signaling labels. We limit the number of issues reported to avoid being perceived as spamming or causing too much workload for project maintainers, and we report 60 issues after this procedure.

At the time of writing, we have received response for 33 reported issues. 16 issues are confirmed as GFIs (see an example in Figure 8) and project maintainers add GFI-signaling labels for these issues, while 17 issues are not considered as GFIs by project maintainers. In other words, our model reaches an estimated precision of $(16+67)/(16+17+67+28) = 64.84\%$ in the real world setting. Although our recommendations still contain false positives (as perceived by the project maintainers), we receive positive responses from the maintainers about our approach:

(1) *Marking more issues as GFI does have the potential of making projects more accessible to new contributors, so I'm intrigued!*[12]
(2) *Thanks for suggesting those. I have gone ahead and added the tag to 3 of the 5. One that I didn't already is a work in progress, the other is a deployment issue. Closing this issue as the tag has been added. Thanks!*[13]
(3) *Thank you. I labelled these but the rest aren't easy: #4456 #4531 #4700.*[14]

Our second goal is to evaluate to what extent are the recommended GFIs actually resolved by newcomers at the time of writing (January 4, 2022). For the 16 issues recommended by us and confirmed later by project maintainers, three issues[15,16,17] (among five resolved) are resolved by a newcomer. One issue[18] has attracted a newcomer but is later resolved by a non-newcomer. Interestingly, for the 17 issues denied by project maintainers, two issues[19,20] (among eight resolved) are also resolved by newcomers, both with only several lines of changes. The two issues offer preliminary evidence that project maintainers may not always be correct in labeling GFIs and an automated recommendation approach adds up more value in helping newcomers onboard.

Summary for RQ3:

RecGFI is helpful in real world scenarios. We report the recommend issues to project maintainers in which 16 are confirmed as GFIs and five have already been resolved by a newcomer.

5 DISCUSSION

Identifying Issues Suitable for Newcomers. Ideally, RecGFI should be able to clearly distinguish between issues suitable / inappropriate for newcomers. Because such ideal ground truth labels are not available, RecGFI is trained from different labels (whether an issue is resolved by a newcomer), which comes with a number of advantages and limitations. First, it works on any project with at least some onboard newcomers even if the project maintainers do not add any GFI-signaling labels to issues. However, although we show that issues resolved by newcomers are less scarce (10.18%-16.26% in Table 1) than issues with GFI-signaling labels (1.93%), it is also possible that issues resolved by non-newcomers are actually suitable for newcomers because it can be hard for newcomers to discover these issues. Such cases can be mitigated if RecGFI is deployed in practice and continuously recommending GFIs to newcomers in a project. We also do not consider the expertise or preference of newcomers during recommendation and we plan to explore the possibility of personalized GFI recommendation in future work.

Filtering of "Not Task" Issues. As demonstrated in Section 4.5, the efficiency of RecGFI is hindered by the fact that a significant portion of issues on GitHub does not correspond to a specific development task [28]. Although this limitation can be mitigated for projects that manage issues with well-defined labels, it may make RecGFI less useful in other projects. Future work can design an approach that automatically learns and filters "not task" issues to make GFI recommendation more effective.

The Threshold k. The threshold used to define newcomers is a parameter that can be flexibly adjusted. If project maintainers tend to define newcomers more strictly, $k = 0$ is a reasonable choice.

[12] https://github.com/facebook/jest/issues/11777
[13] https://github.com/dotnet/machinelearning/issues/5908
[14] https://github.com/sindresorhus/refined-github/issues/4708
[15] https://github.com/freeCodeCamp/freeCodeCamp/issues/43264
[16] https://github.com/facebook/jest/issues/11770
[17] https://github.com/dotnet/machinelearning/issues/5899
[18] https://github.com/sindresorhus/refined-github/issues/4700
[19] https://github.com/sindresorhus/refined-github/issues/4697
[20] https://github.com/facebook/jest/issues/11767

Otherwise, previous work [56] has shown that the distribution of commits are often sparse between newcomers and long-term contributors, so a k value can also be chosen near this boundary, which should be $k = 2$ or $k = 3$ for most projects.

Expertise and Experience in the Newcomer Definition. In this work, we only consider the number of commits for the concept of newcomers and ignores other forms of experiences (e.g., code review, issue reporting, commenting, etc.) because other experiences are either not indicative of expertise in contributing code (e.g., issue reporting, commenting), or are only possible when one already becomes a veteran (e.g., code review). According to Baltes and Diehl [6], knowledge, experience, and skills constitute one's expertise in a project. However, it is non-trivial to infer knowledge and skills from data (which is important for personalized recommendation), so we leave them for future work.

Learning from Issue Text. In RecGFI, we model issue text using readability, keyword frequency, text length, number of images, number of code snippets and number of hyperlinks. Domain-specific information in issue descriptions may be helpful for recognizing GFIs, but we find that straightforward deep neural networks (e.g., LSTM [22]) fail to capture such semantics and cannot outperform our current approach. Therefore, future work can be devoted to approaches that better capture issue semantics.

Generalizability to Other Projects. Theoretically, RecGFI can work on any GitHub project. However, to ensure the quality of our dataset, our evaluation is based solely on projects with many manually labeled GFIs and high popularity. Future work is needed to investigate to what extent can RecGFI perform on projects with different characteristics. If a project lacks historical data (e.g., few onboarded newcomers and few issues), RecGFI may perform less well and its transferability to other projects should also be studied in future work.

6 RELATED WORK

Keeping a good influx of new developers is critical for the survival, long-term success, and continuity of OSS projects [21]. Substantial efforts have been devoted to understanding contributors' motivation, newcomers' barriers, exploring associated factors, and identifying strategies to get people onboard.

Hars et. al [21] identifies two broad types of motivations: internal factors (e.g., intrinsic motivation and altruism) and external rewards (e.g., expected future returns and personal needs). Ye et. al [62] theorize that learning is one of the motivational forces. Von Krogh et al. [30] conclude in a review that developers are motivated by intrinsic, internalized extrinsic, and extrinsic motivations. Krishnamurthy et al. [29] investigate the motivation of peripheral developers and find that the recognition from project stakeholders promotes the participation of peripheral developers.

Newcomers face many barriers when they participate in OSS projects, including both technical (e.g., domain knowledge and programming skills [32, 46]) and non-technical factors (e.g., communication [55]). In a qualitative study, Shibuya et al. [46] find that newcomers usually have difficulties in task selection due to lack of documentation, constraints in contributor license agreement, no technical support, etc. Steinmacher et al. [49] divide 58 difficulties into six groups, e.g., newcomers' characteristics and technical hurdles. Mendez et al. [36] take a new perspective of barriers from tools and infrastructure and find that the barriers are implicated in all six categories of previously established newcomer barrier types.

In view of these barriers, many researchers explore methods and strategies to help newcomers. Zhou [64] calls for research into OSS project communication, modularization, task division, and learning of experts to help newcomer onboarding. Jensen et al. [27] find that timely responses to mails of newcomers can promote them onboard. Zhou et al. [65] find that peers' attention and productivity help newcomers to sustain. Steinmacher et al. [52] propose technical, contribution process, and social behavior guidelines to help newcomers. Considering newcomers' difficulty in finding appropriate tasks [46], many OSS communities try to label issues suitable for newcomers, which inspires researchers' investigations. Steinmacher et al. [50] present a set of strategies identified in literature, interviews, and state-of-the-practice to help newcomers choose suitable task. Tan et al. [56] conduct a first study on the GFI mechanism and find that maintainers usually feel difficult when labelling GFIs and the rationality of the current GFIs is questionable. Alderliesten and Zaidman [1] discover mismatch between GFIs and actual newcomer contributions. Balali et al. [5] identify 7 challenges for maintainers when recommending tasks to newcomers. Horiguchi et al. [23] find that GFIs attract newcomers but are ineffective at retaining them. Existing methods for task selection/recommendation mostly suggest tasks to developers with previous interactions in the project (e.g., Anvik and Murphy [2]). Wang and Sarma [60] develop a tool to help newcomer explore bug reports but their tool is not capable of automated recommendation. In this paper, we fill this gap by proposing a machine learning approach specifically designed for GFI recommendation, which can help relieve maintainers' burden and get newcomers onboard. Recently, a parallel work from Huang et al. [25] propose an automatic approach to characterize and predict GFIs. Our work has several notable differences compared with theirs. First, their approach predicts whether an issue has a GFI label while ours predicts whether an issue will be resolved by a newcomer. Our setup can avoid bias introduced by manual labeling and also works for projects with scarce or no GFI labels. Second, we use the event stream provided by GHTorrent [19] to restore historical issue states. Finally, we build a larger dataset for evaluation, with 10x more projects and two different time points.

7 CONCLUSION

In this paper, we have presented RecGFI, a machine learning approach based on extensive feature engineering and an XGBoost classifier for real GFI recommendation scenarios. We show through experiments that RecGFI achieves high performance, outperforms other alternative methods, and reveals interesting observations on GFI characteristics. We also conduct a field evaluation to prove that RecGFI is able to help project maintainers label GFIs and attract newcomers onboard. In the future, we plan to explore ways to deploy RecGFI (e.g., via GitHub Bots or Plugins) and derive improved approaches for personalized GFI recommendation.

ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China Grant 2018YFB1004201 and the National Natural Science Foundation of China Grant 61825201. We sincerely thank the open-source software developers

who have responded to our issues and the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Jan Willem David Alderliesten and Andy Zaidman. 2021. An Initial Exploration of the "Good First Issue" Label for Newcomer Developers. In *14th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 117–118. https://doi.org/10.1109/CHASE52884.2021.00023

[2] John Anvik and Gail C. Murphy. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.* 20, 3 (2011), 10:1–10:35. https://doi.org/10.1145/2000791.2000794

[3] Deeksha M. Arya, Wenting Wang, Jin L. C. Guo, and Jinghui Cheng. 2019. Analysis and detection of information types of open source software issue discussions. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 454–464. https://doi.org/10.1109/ICSE.2019.00058

[4] Sikha Bagui and Kunqi Li. 2021. Resampling imbalanced data for network intrusion detection datasets. *J. Big Data* 8, 1 (2021), 6. https://doi.org/10.1186/s40537-020-00390-x

[5] Sogol Balali, Umayal Annamalai, Hema Susmita Padala, Bianca Trinkenreich, Marco A. Gerosa, Igor Steinmacher, and Anita Sarma. 2020. Recommending Tasks to Newcomers in OSS Projects: How Do Mentors Handle It?. In *Proceedings of the 16th International Symposium on Open Collaboration* (Virtual conference, Spain) *(OpenSym 2020)*. Association for Computing Machinery, New York, NY, USA, Article 7, 14 pages. https://doi.org/10.1145/3412569.3412571

[6] Sebastian Baltes and Stephan Diehl. 2018. Towards a theory of software development expertise. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 187–200. https://doi.org/10.1145/3236024.3236061

[7] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, Mary Jean Harrold and Gail C. Murphy (Eds.). ACM, 308–318. https://doi.org/10.1145/1453101.1453146

[8] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 785–794. https://doi.org/10.1145/2939672.2939785

[9] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. ACM, 186–196. https://doi.org/10.1145/3106237.3106246

[10] Sanjay Kumar Dubey, Ajay Rana, and Yajnaseni Dash. 2012. Maintainability prediction of object-oriented software system by multilayer perceptron model. *ACM SIGSOFT Softw. Eng. Notes* 37, 5 (2012), 1–4. https://doi.org/10.1145/2347696.2347703

[11] Olive Jean Dunn. 1961. Multiple comparisons among means. *J. Amer. Statist. Assoc.* 56, 293 (1961), 52–64. https://doi.org/10.1080/01621459.1961.10482090

[12] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E. Hassan. 2020. Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. *IEEE Trans. Software Eng.* 46, 5 (2020), 495–525. https://doi.org/10.1109/TSE.2018.2864217

[13] GitHub, Inc. 2021. *Autolinked references and URLs.* Retrieved August 27, 2021 from https://docs.github.com/en/github/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls

[14] GitHub, Inc. 2021. *Encouraging helpful contributions to your project with labels.* Retrieved June 17, 2021 from https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions/encouraging-helpful-contributions-to-your-project-with-labels

[15] GitHub, Inc. 2021. *GitHub REST API.* Retrieved August 23, 2021 from https://docs.github.com/en/rest

[16] GitHub, Inc. 2021. *Issue event types.* Retrieved August 27, 2021 from https://docs.github.com/en/developers/webhooks-and-events/events/issue-event-types

[17] GitHub, Inc. 2021. *Linking a pull request to an issue.* Retrieved August 27, 2021 from https://docs.github.com/en/issues/tracking-your-work-with-issues/linking-a-pull-request-to-an-issue

[18] GitHub, Inc. 2021. *Setting up your project for healthy contributions.* Retrieved June 17, 2021 from https://docs.github.com/en/communities/setting-up-your-project-for-healthy-contributions

[19] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 233–236.

[20] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.

[21] A. Hars and Shaosong Ou. 2001. Working for free? Motivations of participating in open source projects. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. 9 pp.–. https://doi.org/10.1109/HICSS.2001.927045

[22] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[23] Hyuga Horiguchi, Itsuki Omori, and Masao Ohira. 2021. Onboarding to Open Source Projects with Good First Issues: A Preliminary Analysis. In *28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021*. IEEE, 501–505. https://doi.org/10.1109/SANER50967.2021.00054

[24] Yonghui Huang, Daniel Alencar da Costa, Feng Zhang, and Ying Zou. 2019. An empirical study on the issue reports with questions raised during the issue resolving process. *Empir. Softw. Eng.* 24, 2 (2019), 718–750. https://doi.org/10.1007/s10664-018-9636-3

[25] Yuekai Huang, Junjie Wang, Song Wang, Zhe Liu, Dandan Wang, and Qing Wang. 2021. Characterizing and Predicting Good First Issues. In *ESEM '21: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, October 11-15, 2021*, Filippo Lanubile, Marcos Kalinowski, and Maria Teresa Baldassarre (Eds.). ACM, 13:1–13:12. https://doi.org/10.1145/3475716.3475789

[26] Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belen Rolandi, Alexandre Bergel, and Jordi Cabot. 2015. GiLA: GitHub label analyzer. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015*, Yann-Gaël Guéhéneuc, Bram Adams, and Alexander Serebrenik (Eds.). IEEE Computer Society, 479–483. https://doi.org/10.1109/SANER.2015.7081860

[27] Carlos Jensen, Scott King, and Victor Kuechler. 2011. Joining Free/Open Source Software Communities: An Analysis of Newbies' First Interactions on Project Mailing Lists. In *44th Hawaii International International Conference on Systems Science (HICSS-44 2011), Proceedings, 4-7 January 2011, Koloa, Kauai, HI, USA*. IEEE Computer Society, 1–10. https://doi.org/10.1109/HICSS.2011.264

[28] 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

[29] Rajiv Krishnamurthy, Varghese Jacob, Suresh Radhakrishnan, and Kutsal Dogan. 2016. Peripheral Developer Participation in Open Source Projects: An Empirical Analysis. *ACM Trans. Manage. Inf. Syst.* 6, 4, Article 14 (Jan. 2016), 31 pages. https://doi.org/10.1145/2820618

[30] Georg Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin Wallin. 2012. Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Quarterly* forthcoming (06 2012). https://doi.org/10.2307/41703471

[31] Karim R Lakhani and Robert G Wolf. 2003. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Open Source Software Projects (September 2003)* (2003). https://dx.doi.org/10.2139/ssrn.443040

[32] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE / ACM, 187–197. https://doi.org/10.1109/ICSE.2017.25

[33] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics* (1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[34] Lionel Marks, Ying Zou, and Ahmed E. Hassan. 2011. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering, PROMISE 2011, Banff, Alberta, Canada, September 20-21, 2011*, Tim Menzies (Ed.). ACM, 11. https://doi.org/10.1145/2020390.2020401

[35] Francisco Melo. 2013. *Area under the ROC Curve.* Springer New York, New York, NY, 38–39. https://doi.org/10.1007/978-1-4419-9863-7_209

[36] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. 2018. Open Source Barriers to Entry, Revisited: A Sociotechnical Perspective. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 1004–1015. https://doi.org/10.1145/3180155.3180241

[37] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11, 3 (2002), 309–346. https://doi.org/10.1145/567793.567795

[38] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for engineered software projects. *Empir. Softw. Eng.* 22, 6 (2017), 3219–3253. https://doi.org/10.1007/s10664-017-9512-6

[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830. http://dl.acm.org/citation.cfm?id=2078195

[40] Eric Raymond. 1999. The cathedral and the bazaar. *Knowledge, Technology & Policy* 12, 3 (1999), 23–49.

[41] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 1135–1144. https://doi.org/10.1145/2939672.2939778

[42] S. Abijah Roseline and S. Geetha. 2018. Intelligent Malware Detection using Oblique Random Forest Paradigm. In *2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018, Bangalore, India, September 19-22, 2018*. IEEE, 330–336. https://doi.org/10.1109/ICACCI.2018.8554903

[43] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *J. Syst. Softw.* 158 (2019). https://doi.org/10.1016/j.jss.2019.110406

[44] Gerard Salton and Chris Buckley. 1988. Term-Weighting Approaches in Automatic Text Retrieval. *Inf. Process. Manag.* 24, 5 (1988), 513–523. https://doi.org/10.1016/0306-4573(88)90021-0

[45] Benedikt Schifferer, Chris Deotte, and Even Oldridge. 2020. Tutorial: Feature Engineering for Recommender Systems. In *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, Rodrygo L. T. Santos, Leandro Balby Marinho, Elizabeth M. Daly, Li Chen, Kim Falk, Noam Koenigstein, and Edleno Silva de Moura (Eds.). ACM, 754–755. https://doi.org/10.1145/3383313.3411543

[46] Bianca Shibuya and Tetsuo Tamai. 2009. Understanding the process of participating in open source communities. In *2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. IEEE, 1–6. https://doi.org/10.1109/FLOSS.2009.5071352

[47] Dmitry Smolyakov, Alexander Korotin, Pavel Erofeev, Artem Papanov, and Evgeny Burnaev. 2018. Meta-learning for resampling recommendation systems. In *Eleventh International Conference on Machine Vision, ICMV 2018, Munich, Germany, 1-3 November 2018 (SPIE Proceedings, Vol. 11041)*, Antanas Verikas, Dmitry P. Nikolaev, Petia Radeva, and Jianhong Zhou (Eds.). SPIE, 110411S. https://doi.org/10.1117/12.2523103

[48] Christoph Stanik, Lloyd Montgomery, Daniel Martens, Davide Fucci, and Walid Maalej. 2018. 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 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 172–182. https://doi.org/10.1109/ICSME.2018.00027

[49] Igor Steinmacher, Ana Paula Chaves, Tayana Uchôa Conte, and Marco Aurélio Gerosa. 2014. Preliminary Empirical Identification of Barriers Faced by Newcomers to Open Source Software Projects. In *2014 Brazilian Symposium on Software Engineering, Maceió, Brazil, September 28 - October 3, 2014*. IEEE Computer Society, 51–60. https://doi.org/10.1109/SBES.2014.9

[50] 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 *48th Hawaii International Conference on System Sciences, HICSS 2015, Kauai, Hawaii, USA, January 5-8, 2015*. IEEE Computer Society, 5299–5308. https://doi.org/10.1109/HICSS.2015.624

[51] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco Aurélio Gerosa. 2018. Almost There: A Study on Quasi-Contributors in Open-Source Software Projects. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 256–266. https://doi.org/10.1145/3180155.3180208

[52] Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2019. Let Me In: Guidelines for the Successful Onboarding of Newcomers to Open Source Projects. *IEEE Softw.* 36, 4 (2019), 41–49. https://doi.org/10.1109/MS.2018.110162131

[53] Mervyn Stone. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)* 36, 2 (1974), 111–133. https://doi.org/10.1111/j.2517-6161.1974.tb00994.x

[54] Margaret-Anne D. Storey, Jody Ryall, Janice Singer, Del Myers, Li-Te Cheng, and Michael J. Muller. 2009. How Software Developers Use Tagging to Support Reminding and Refinding. *IEEE Trans. Software Eng.* 35, 4 (2009), 470–483. https://doi.org/10.1109/TSE.2009.15

[55] Xin Tan and Minghui Zhou. 2019. How to Communicate when Submitting Patches: An Empirical Study of the Linux Kernel. *Proc. ACM Hum. Comput. Interact.* 3, CSCW (2019), 108:1–108:26. https://doi.org/10.1145/3359210

[56] Xin Tan, Minghui Zhou, and Zeyu Sun. 2020. A first look at good first issues on GitHub. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 398–409. https://doi.org/10.1145/3368089.3409746

[57] Feifei Tu, Jiaxin Zhu, Qimu Zheng, and Minghui Zhou. 2018. Be careful of when: an empirical study on time-related misuse of issue tracking data. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 307–318. https://doi.org/10.1145/3236024.3236054

[58] Marat Valiev, Bogdan Vasilescu, and James D. Herbsleb. 2018. Ecosystem-level determinants of sustained activity in open-source projects: a case study of the PyPI ecosystem. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 644–655. https://doi.org/10.1145/3236024.3236062

[59] Lev Vygotsky. 1978. Interaction between learning and development. *Readings on the development of children* 23, 3 (1978), 34–41.

[60] 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, CHASE 2011, Waikiki, Honolulu, HI, USA, May 21, 2011*, Marcelo Cataldo, Cleidson R. B. de Souza, Yvonne Dittrich, Rashina Hoda, and Helen Sharp (Eds.). ACM, 76–79. https://doi.org/10.1145/1984642.1984661

[61] Mohamad Yazdaninia, David Lo, and Ashkan Sami. 2021. Characterization and Prediction of Questions without Accepted Answers on Stack Overflow. In *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 59–70. https://doi.org/10.1109/ICPC52881.2021.00015

[62] Yunwen Ye and Kouichi Kishida. 2003. Toward an Understanding of the Motivation Open Source Software Developers. In *Proceedings of the 25th International Conference on Software Engineering* (Portland, Oregon) *(ICSE '03)*. IEEE Computer Society, USA, 419–429.

[63] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Real-time incident prediction for online service systems. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 315–326. https://doi.org/10.1145/3368089.3409672

[64] Minghui Zhou. 2019. *Onboarding and Retaining of Contributors in FLOSS Ecosystem*. Springer Singapore, Singapore, 107–117. https://doi.org/10.1007/978-981-13-7099-1_7

[65] Minghui Zhou and Audris Mockus. 2012. What make long term contributors: Willingness and opportunity in OSS community. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE Computer Society, 518–528. https://doi.org/10.1109/ICSE.2012.6227164

[66] Minghui Zhou and Audris Mockus. 2015. Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. *IEEE Trans. Software Eng.* 41, 1 (2015), 82–99. https://doi.org/10.1109/TSE.2014.2349496

[67] Minghui Zhou, Audris Mockus, Xiujuan Ma, Lu Zhang, and Hong Mei. 2016. Inflow and Retention in OSS Communities with Commercial Involvement: A Case Study of Three Hybrid Projects. *ACM Trans. Softw. Eng. Methodol.* 25, 2 (2016), 13:1–13:29. https://doi.org/10.1145/2876443