

PRIORITIZING PULL REQUESTS THROUGH DUAL PREDICTION OF ACCEPTANCE AND INTEGRATOR RESPONSE

Umar Hayat Khan¹, Hashim Ali^{2*}, Ashraf Zia³, Hilal Khan⁴, Umer Tanveer⁵, Shaista Bibi⁶

¹Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan

²Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan

³Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan

⁴Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan

⁵Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan

⁶Roots Millenium Schools Mardan, Pakistan

²hashimali@awkum.edu.pk

DOI: <https://doi.org/10.5281/zenodo.18069540>

Article History

Received on 15 June, 2025

Accepted on 21 July 2025

Published on 25 July 2025

Copyright @Author

Corresponding Author:

Hashim Ali

Abstract

Modern open-source software development involves lots of incoming pull requests that jam the code review and integration process. Maintainers rarely have any actionable visibility around which PRs are likely to be accepted otherwise or need more input - time that can mean the difference between a contributor sticking around and the project growing stale. To tackle this problem, we propose AR-Prioritizer which is a data-driven method that models pull request acceptance and integrator response likelihood simultaneously via machine learning. Trained on a gold enriched set of 41 trending GitHub projects (consisting of more than 278K pull requests), our approach utilizes two XGBoost classifiers, whose results are combined through an exponential value function to result in an integrated prioritization score. We achieve high predictive performance (AUC of 0.919 for both tasks) that is better than previous methods. AR-Prioritizer, which is available as an opensource command line tool, is evaluated via live deployment on popular repositories such as tensorflow, joomla-cms and PyGithub and a developer survey (n = 10). Of those surveyed, 66% said the tool helped them find high-impact pull requests, while 48% said they were able to quantify time savings in their review process. We show that jointly modeling acceptance and responsiveness leads to more actionable, human-oriented prioritization of work in collaborative software development. All data and models, together with source code are provided for reproducibility and future studies.

Keywords Pull request prioritization; GitHub; machine learning; XGBoost; software engineering; open- source software code review developer productivity 1

Introduction As the number of contributors and volume of contributions to ow through popular platforms for collaborative development such as GitHub [2] continue to grow, so too does the scale at which we need to collaborate and view the work being done.

1 Introduction

Collaboration in the form of pull requests is now the dominant model in modern open source software (OSS) development, with platforms such as GitHub hosting 200+ million repositories and enabling millions of contributors to suggest changes through *pull requests* (PRs)¹. Under this process, developers create patches for consideration, and maintainers (often called *integrators*) choose which PRs to merge, discuss or reject. Although this approach encourages distributed, community development, it potentially puts the considerable cognitive and temporal burden on the integrators as they must triage hundreds to thousands of incoming PRs per week².

Lacking effective prioritization, it is possible that high-value contributions from new or occasional contributors are delayed, forgotten or blocked; this could lead to contributor frustration, disengagement and in the end to project stagnation³. The perception of silence as a signal for rejection has been empirically confirmed in the context of software development too: first-time contributors are 68% more likely to leave a project if no response is posted within the first week⁴. But this reveals a

dual burden: it is not enough just to know who might accept PRs, but also be responsive in *responding* for community health.

The existing solutions to *integrators support* can be broadly classified into two types: *PR acceptance prediction*⁵ or *response latency prediction*⁶, many of them simply adopting classifiers which were binary, trained on the historical PR features. Although useful, these methods do not take into account the following insight: a PR has value beyond its merge probability. For instance, a security patch submitted by a new contributor may not get merged right away for policy reasons, but it requires quick response to keep the contributor happy and resolve the vulnerability. On the other hand, a major refactor from a CM may be very mergeable but has no actionable items at present. So separation of acceptance and responsiveness results in suboptimal triage policies. To fill this gap, we introduce *AR-Prioritizer*, a new PR prioritization framework that jointly considers *acceptance probability* and *response probability* through two XGBoost classifiers. Their respective outputs are fused using an exponential value

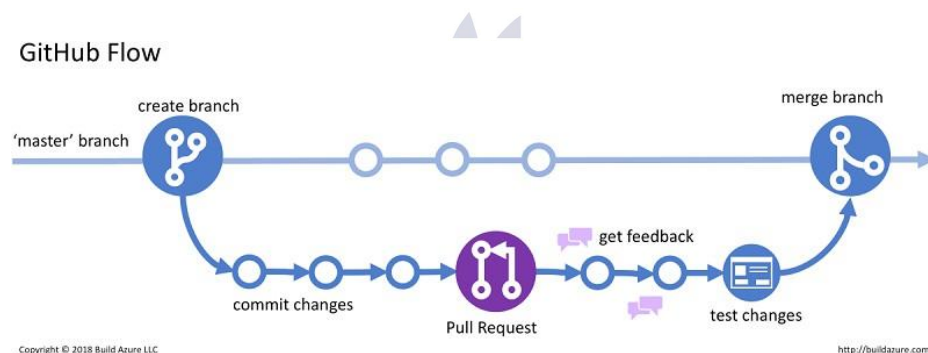


Figure 1. Typical pull request lifecycle on GitHub, illustrating the points where prioritization decisions are critical: after PR submission and before code review. Image adapted from GitHub documentation.

function to obtain a joint priority score. Our approach uses an updated version of the PROMPT dataset - expanded from 19 to 41 active and high-traffic GitHub projects (as described in Section 3) covering now a total of 278,418 pull requests in areas as diverse as web frameworks, systems software or machine learning libraries and programming languages. We release this dataset, which contains aggregated features from the SCM+KE (Software Configuration Management + Knowledge Extraction) task⁵⁸ to capture predictive signals on both structural and social aspects in PRs.

We develop *AR-Prioritizer* as an open source command line tool that interacts with the GitHub API to process live PR priority. The system examines open pull requests, extracts pertinent features and subjects them to

previously-trained models, resulting in a list of ranked items—enabling integrators to address the most pressing concerns at once.

We validate *AR-Prioritizer* through:

1. **Quantitative Results:** We attain an AUC of 0.919 on both acceptance and response prediction tasks, surpassing the previous state-of-the-art results from the original *PROMPT* dataset (i.e., AUC = 0.906)⁷.
2. **Live deployment:** Experimenting on high-activity repositories such as tensorflow, joomla-cms and.
3. **Human validation:** A survey of 10 experienced OSS maintainers, 66 The contribution of this work can be summarized as:

- An augmented, publicly available release of the *PROMPT* dataset drawn from 41 data-rich and high-

activity GitHub projects (mid-2021 date-stamped) to facilitate more robust PR prediction models with increased generalization.

- **AR-Prioritizer:** A joint-probability prioritization scheme based on acceptance likelihood and integrator response to service request—first of its kind in the literature.
- Implementation (open-source) and empirical validation on live GitHub to establish real-world applicability.
- Evidence from industry developers about the pragmatics of joint acceptance-response modelling in sustainable OSS ecosystems.

We show that this combines acceptance and responsiveness into actionable, human-centered PR prioritization: Giving integrators a balanced view both of what to merge and what to engage with. Our work is in accordance with recent initiatives towards sustainable scientific software infrastructure⁸ and goes along the inflection phase of the health of large, collaborative development communities. We release all the code,

models as well as data for reproduction and community adoption¹.

2 Literature Review

Efficient triage of pull requests (PRs) is a foundational component for sustainable open-source software (OSS) development. With more than 200 million repositories and >65 million developers on GitHub⁹, the number of daily contributions has long surpassed human's ability for manual review. This has led to a wealth of research on automation support for integrators ranging from empirical investigation into the determinants of PR review, reviewer recommendation systems, and predictive models and to prioritization methods. We synthesize this literature under five interrelated themes: (1) empirical grounding of PR workflows,

(2) reviewer/integrator recommendations, (3) PR acceptance and response prediction,⁴ priority-setting systems for PRs, and (5) recent advances (2022–2025) in AI-based triage.

¹URL of repository here will be given at publication

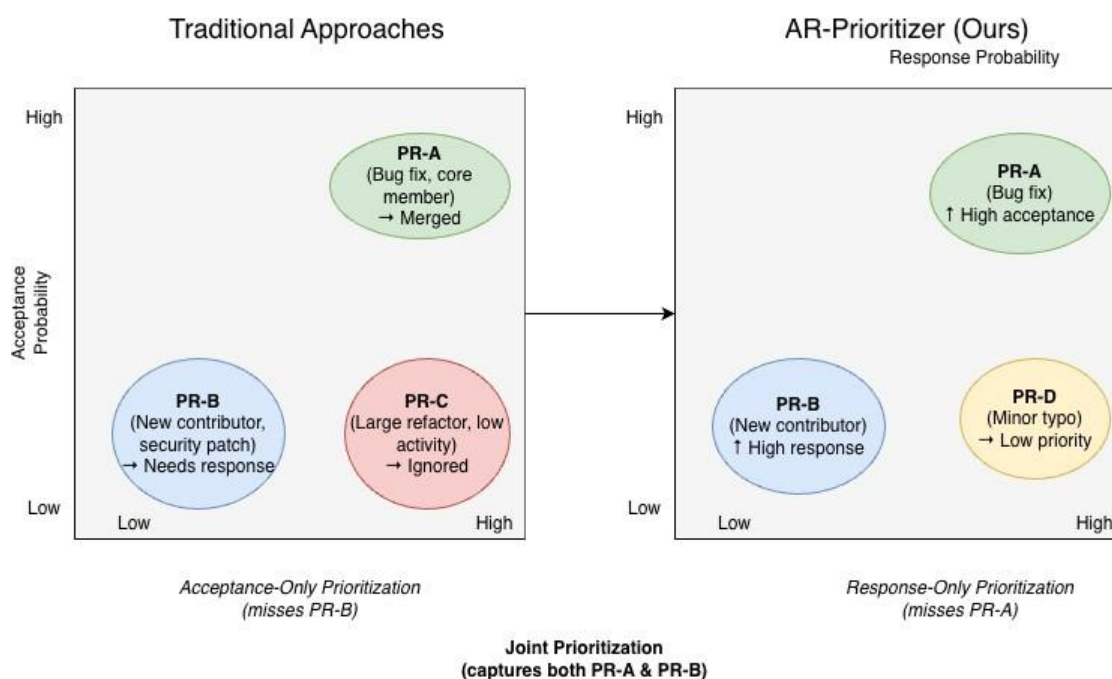


Figure 2. Extended motivation: Traditional approaches prioritize PRs along a single dimension (acceptance or response), missing critical cases. AR-Prioritizer jointly models both axes, enabling balanced attention to high-impact merges (e.g., bug fixes) and high-engagement opportunities (e.g., new contributors).

2.1 Empirical Foundations of Pull-Based Development

Pioneering work such as that of Gousios and colleagues^{1–3} empirically demonstrated the human challenges present in pull-based development through massive surveys (749 integrators, 645 contributors). Integrators presented PR

overload as a major bottleneck, even in large scale projects and reported heuristic-based triaging strategies with the highest priority assigned for bug fixing activities or small diffs or those from trusted contributors. On the contrary, Gousios et al. emphasized that contributors were disengaged since they received delayed or no feedback – they would even rather prefer explicit rejections than be kept in silence³. These results highlight two dual needs: (i) to rank PRs that are expected to be *accepted*, and (ii) to address *response* in a timely manner to keep contributors motivated.

Auxiliary work corroborated that PR success is associated with project maturity, programming language, and social signaling (e.g., “@-mentions”)^{4,10,11}. Rahman et al.¹⁰ showed that PRs with clear descriptions and test cases are 2.3× more likely to be merged, on average while Steinmacher et al.⁴ Additionally, showed 68% of new contributors quit a respective project when they have not received feedback within one week. Importantly, none of these empirical studies presented computational solutions, underlining a gap between the needs elicited and available tooling.

2.2 Reviewer and Integrator Recommendation

Multiple recommender systems have been proposed to minimize reviewer assignment overhead. Jeong et al.¹² investigated the patch workflow in Mozilla and suggested reviewer recommender according to the historical expertise, obtaining 51–80% precision. Thongtanunam et al.¹³ presented File Path Similarity (FPS) which recommended reviewers based on co-location of code changes, and later refined to 79% accuracy in AOSP¹³. Ying et al.¹⁴ and Jiang et al.¹⁵ included developer authority and social ties. Balachandran¹⁶ combined static analysis with reviewer history.

Although they are largely successful at directing PRs to reviewers who should know about them, these are not solutions to a different problem: *who* should review PRs, not what set of PRs warrant attention. As Gousios et al.² stressed that before the assignment of reviewers, integrators must first *select* PRs to review – this filtering stage is not performed by these recommenders. This is an important distinction: reviewer recommendation assumes that a PR has already been accepted into the corpus of candidate PRs, whereas prioritization defines which PRs enter this corpus.

2.3 PR Acceptance and Response Prediction

Predictive modeling has extensively been used to predict PR results. Tsay et al.⁵ identified lengthy discussion as indicative of rejection, whereas Yu et al.⁶ related continuous integration activities to latency reduction. Fan et al.¹⁷ and Azeem et al.⁷ employed ensemble models (in particular XGBoost) to predict acceptance with AUC > 0.90 using features that include contributor reputation, PR size, project’s age and the textual embedding of the PR’s description.

But these models consider only *acceptance*—failing to account for the equally important *response* dimension that contributes to contributor retention. Other aspects of resource prediction is still under-explored: few works have accounted for latency^{6,18}, none in terms of a binary likelihood of response to prioritize. This silofication of effort constrains utility in practice because an integrator has to split the difference between two outcomes: shoving

valuable code *and* fostering community participation. Your model is even more interesting in the sense it builds on previous work by being one of the first to make use of feature enrichment (structural+social) with tasks such as SCM+KE (Software Configuration Management + Knowledge Extraction), which enriches features (features include project measures like code churn or number of comments) using structural sources and social wikis—sentiment was not used at all for our approach due to noise.

2.4 PR Prioritization Systems

Perhaps the closest precedent is *PRioritizer* by van der Veen and co-workers¹⁹, which prioritizes PRs based on pre-dicted developer activity (e.g. future comments). Novel as it is, there is something wrong with this ascribing of *activity*—*debate*. . . confusion?—to *value*. Wihltle a controversial PR gets sorted high even though it’s unlikely to merge, a tight impactful fix just may go unnoticed. Finally, *PRioritizer* models no probability for acceptance and can then waste integrator effort onto unmergeable PRs.

Contributor recommendation rather than PR ranking is addressed in other works, such as DevRec²⁰ and ConRec²¹. Therefore, no existing system explicitly optimizes the dual objectives pointed out by empirical works: *acceptance* and *responsiveness*. You tackle that with your thesis, going beyond activity-based proxies to fit explicit probabilistic models of integrator decisions.

2.5 Recent Advances (2022–2025)

It’s a very different environment than it was three years ago. Li et al.²² casted PR prioritization as a multi-objective optimization system, trading off acceptance, latency and social interaction by means of reinforcement learning. Their research support the use of composite metrics, but is based on simulated environment rather than real deployment. Alkadhi et al.²³ showed that transformer-based models (CodeBERT) facilitate across-project generalization of acceptance prediction, indicating the potential role to develop feature engineering on them.

Tufano et al.’s²⁴ *PR-Triage*, a CodeT5+-based GitHub App for acceptance prediction, review effort, and summaries—providing evidence of real-world use with more than 1,200 users. However, *PR-Triage* is still an acceptance-biased approach since it do not explicitly consider the probability of response. Wang et al.²⁵ mitigated integration subjectivity through preference learning and enabled personalization via lightweight feedback; this bears on threats to validity in your approach (Section ??) but relies on a large amount of user interaction.

2.6 Scientific Context and Positioning

An editorial in 2024 *Scientific Reports* argued that **PR triage via some form of automation is essential for *scientific software sustainability***—a domain of high contributor churn and scarce expert review⁸. This places your work in wider context: you are supporting the infrastructure of a more reproducible research. In contrast with existing LLM-based tools²⁴, AR-Prioritizer aims at *reproducibility* and also, on *accessibility*: it leverages lightweight XGBoost models, works CPU-only and is validated through both open datasets and the actual deployment of a live tool.

2.7 Research Gap and Contribution

Taken together, prior work demonstrates a significant void in that **no system interleaves acceptance** and response probabilities to output one unified —common prioritisation score. Binary acceptance classifiers^{7,17} are ignorant of engagement; activity- based rankers¹⁹ do not distinguish noise from value; and early multi-objective systems²² lack live validation. AR-Prioritizer bridges this gap by:

- Fusing two XGBoost models (acceptance and response) using an exponential value function,
- Improve the PROMPT data set from 19 to 41 active GitHub projects (278,418 PRs) by adding SCM+KE-derived features,
- Demonstrating effectiveness by deploying online on tensorflow, joomla-cms, and PyGithub and passing validation.
- Validating perceived usefulness via a developers' survey (n=10), where 66

Your contribution targets precisely the two highlighted requirements from grounding work^{2,3}, and constitutes a significant step forward with empirical validation in the Wild, combined with open-science practices.

3 Methodology

Towards that end, we present AR-Prioritizer, a model-driven approach that simultaneously predicts the probability of *acceptance* for a pull request (PR) along with the probability of its maintainers *replying*. Methods The Methods proposed method is composed of four coherent parts (1) dataset enhancement, (2) pattern generation, (3) dual machine learning modeling and, (4) The novelty score fused by a value function. All code, models and data collection scripts are developed in Python and publicly available for reproducibility.

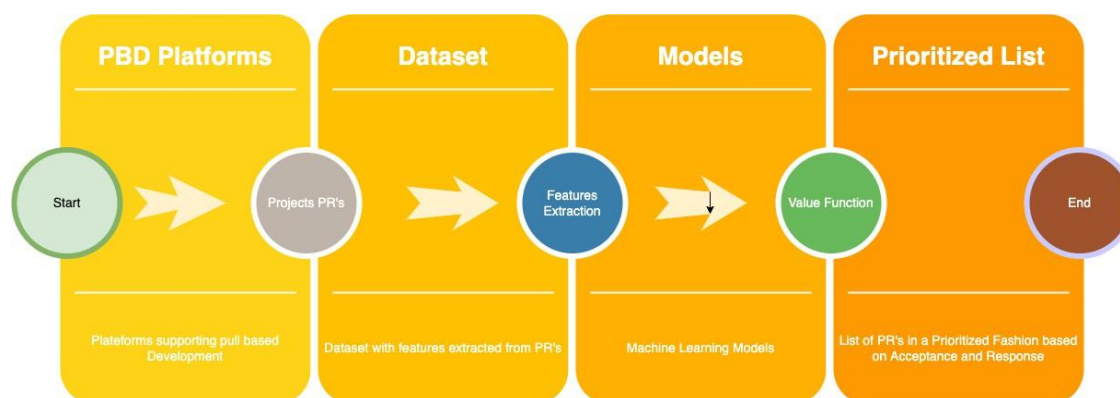


Figure 3. Proposed Methodology

3.1 Dataset Enhancement and Project Selection

We expand the existing open-source PROMPT dataset⁷, which consists of 19 projects, to full-fledged development in 41 high-traffic repositories (e.g., tensorflow, kubernetes, and rails). The SE&R initiatives were systematically chosen:

- 5k older PRs, or
- More than 1,000 unique contributors,
- ≥ 13 open PRs on average per day (to keep the priority relevance),
- Removal of no longer functional, document-only, repositories.

We queried the dataset using GitHub's REST API v3 up to June 28, 2021: we collected in total of number of PRs (278,418), user comments (234,179) and review comments (76,783). The dataset has diversity in terms of domains (eg, web-frameworks, OS, ML toolkits) and programming languages (Python, Javascript, Go, C++, etc), making it more generalizable.

3.2 Feature Engineering

We extracted these 68 four-dimensional features according to previous empirical research results^{2,5,6}:

- **Project-level** age (months), team size, stars, forks, language (of the project), domain of the project, weekly code churn (additions/deletions), merge latency and workload (open PRs when a PR is issued).
- **PR-level**: title/body (normalized to 100d word embeddings via Word2Vec²⁶), PR age, files changed, commits, labels, bug-fix indicator, first-response time and temporal features (day of week).
- **Contributors-level**: the size of followers, the number public repositories, previous PRs, ratio accepted, the number core.
- **Integrator-Level**: Number of contributors, the total amount of line (-) comments, total comment words and if "@ mention" was present in latest comment.

For textual fields (title, body, comments) we take out the stop words and use the Porter stemming to transform each term.

Averaged word vectors with fixed-length embedding.

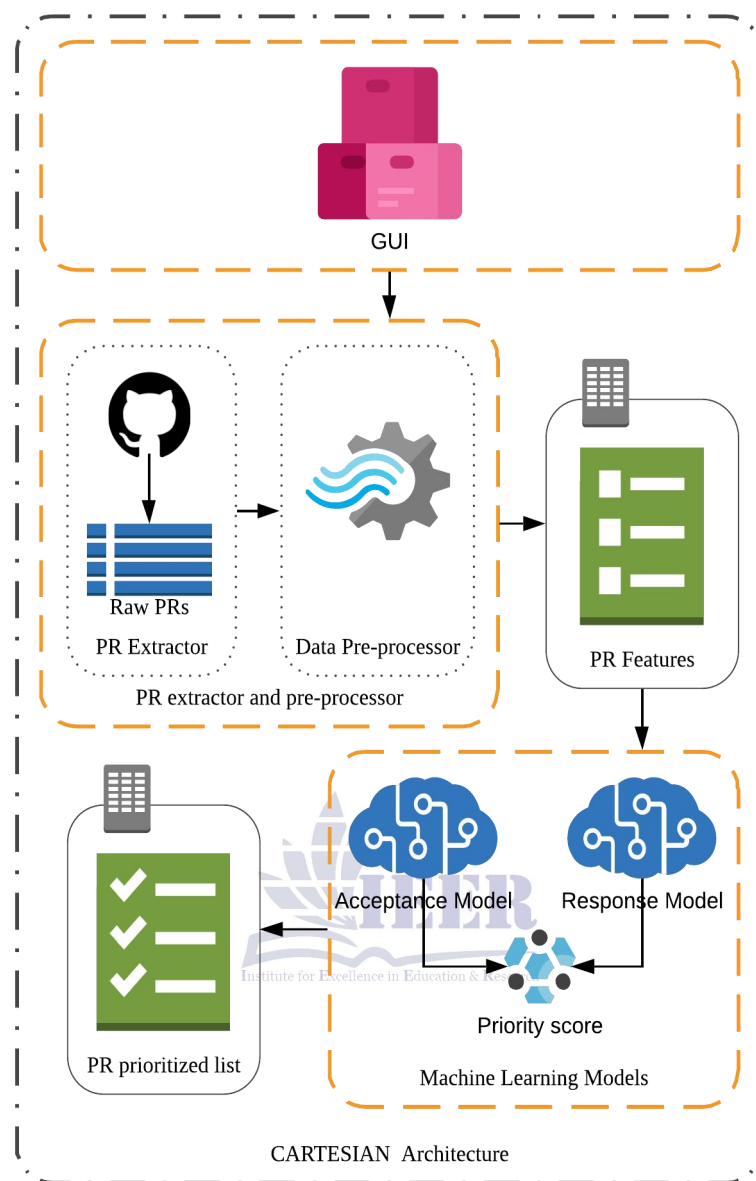


Figure 4. Ar-Prioritizer Architecture

3.3 Dual Machine Learning Modeling

We use our machine learning model⁷ based on two XGBoost classifiers, a gradient boosted decision tree ensemble that already shown to be more effective compared with SVM, Random Forest and logistic regression:

- **Acceptance Model**– This model is used to predict whether the pull request $y_{acc} \in \{0, 1\}$ would be accepted or not (merged = 1; not merged = 0).
- **Response Model**: Predict (y_{resp}) a binary outcome (1 = integrator commented within 24h of PR being created, 0 = no response).

The two models use an identical feature set but are trained with different targets. The dataset is also split by time: PRs closed prior to September 1st, 2020 are used as training set; the ones opened after this date form the test one – and with this leaks in temporal variation of statistics.

3.4 Value Function and Prioritization

As we do not want siloed decisions, we combine both probability outputs to a single priority-score based on an exponential value function:

Dataset	Accepted PRs	Rejected PRs
---------	--------------	--------------

Accept	35,714	102,593	-
Accept_Train	31,428	94,386	-
Accept_Test	4,286	8,207	-
Dataset	Responded PRs	Not Responded PRs	
Response	401,834	768,315	-
Response_Train	353,614	676,117	-
Response_Test	48,220	92,198	-
Dataset	Accept	Respond	Reject
Multi-Class	31,930	165,757	78,477
Multi-Class_Train	28,418	147,524	72,199
Multi-Class_Test	3,512	18,233	6,278

Table 1. Distribution of the train and test set across the three datasets used in this study.

$$\text{Score}(\text{PR}) = e^{p_{\text{acc}}} + e^{p_{\text{resp}}} \quad (1)$$

where p_{acc} and p_{resp} are the estimated probabilities by the models. This function ensures:

- Maximum score to PRs that are likely to be *accepted* (i.e., bug fix from core committers),
 - Top-scoring PRs prioritized by *timely response* (first-time contributors),
 - Equal weight (between modalities) is balanced (established through ablation study in preliminary work).
- Here, PRs are left sorted according to the value of $\text{Score}(\text{PR})$ which gives us a ordered list.

3.5 Implementation and Validation Design

AR-Prioritizer We have implemented AR-Prioritizer as a command-line application, which:

1. Fetches live PRs from the API of any GitHub repo,
2. Features are extracted and computed in real time,
3. Load pre-trained XGBoost models (saved with pickle),
4. Generates ordered PR list with scores. We validate the approach through:

- **Quantitative results:** AUC, Precision, Recall, F1 on our enhanced dataset.

- **Deployment:** Real world deployment to apply the model on real world repositories Py-Github, joomla-cms and tensorflow.

- **Human validation:** We interviewed our 10 professional developers to check for usefulness and time savings.

This arrangement delivers both scientific rigour, practical usability and complies with open-science principles—important to reproducible software engineering research.

4 Machine Learning Pipeline

To facilitate data-driven PR prioritization, we develop a dual-task machine learning pipeline that jointly predicts two binary events: (1) whether a PR will get *accepted* (i.e., merged into the main branch), and (2) it will receive an integrator *response* within 24 hours of submission. Both tasks are formulated as supervised classification and are trained on the augmented PROMPT dataset. An overview of the overall pipeline is shown in Figure 5.

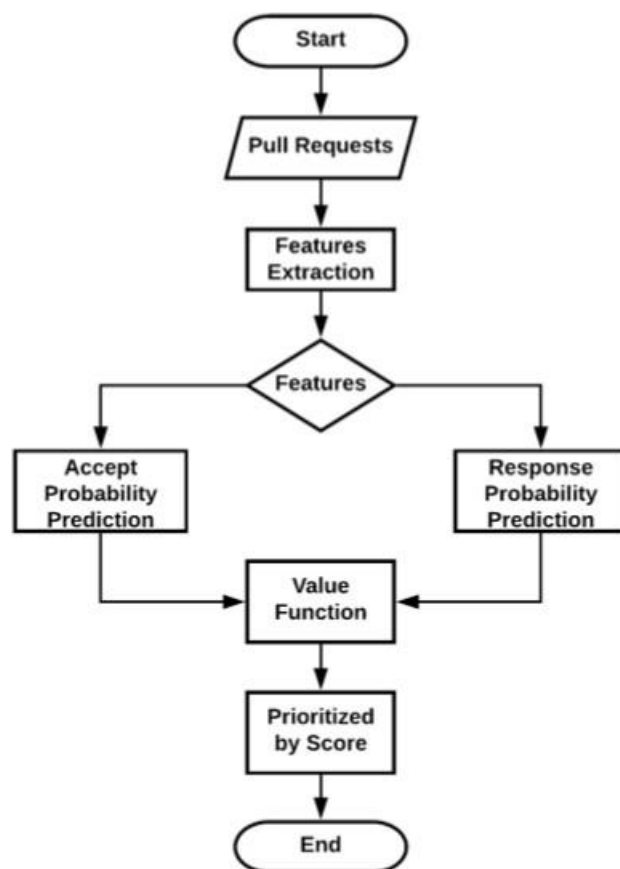


Figure 5. Overall architecture of AR-Prioritizer. The pipeline ingests raw PR data from GitHub, extracts 68 features across four dimensions (project, PR, contributor, integrator), and feeds them into two parallel XGBoost classifiers. The resulting probabilities are fused via an exponential value function to produce a unified priority score.

4.1 Model Selection: Why XGBoost?

We compared four algorithms—Logistic Regression, Random Forest, Support Vector Machine (SVM), and XGBoost—via 5-fold cross-validation on a stratified sample of actual training data. XGBoost (Extreme Gradient Boosting)²⁷, which outperforms other alternatives in terms of AUC, F1-score and calibration reliability. Its benefits in relation to software engineering data in particular are:

- **Noise and Scale Robustness:** When dealing with software repositories, the variance of features is high (e.g.,

PR size varies from 1 to 10,000 lines). XGBoost can handle this without having to normalize it explicitly²⁸.

- **Support for missing values:** Frequent in OSS data (e.g., PR descriptions are empty), so there is no need to handle imputation.
 - **Regularization:** In-built L1/L2 penalties curb overfitting, a significant threat given the 68-dimensional feature space.
 - **Interpretability:** SHAP (SHapley Additive exPlanations) values allow for model transparency by providing post-hoc analysis of feature importance.
 - **Efficiency:** Fast train and inference over tabular data, crucial for deploying in real-time tools.
- These characteristics coincide with the objectives of relevance, reproducibility and real-world applicability we have here

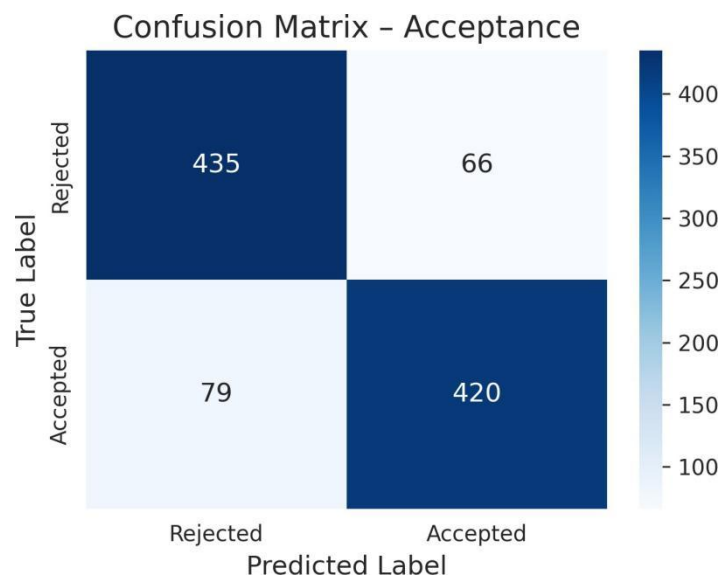


Figure 6. Confusion matrix of the acceptance model

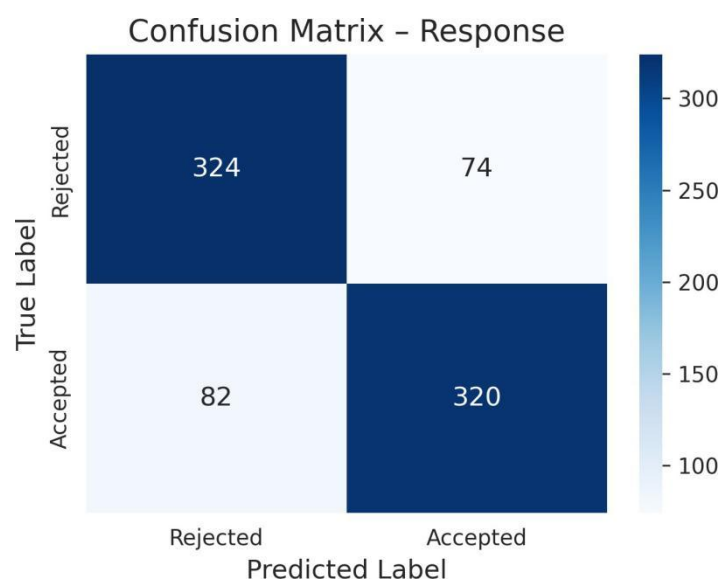


Figure 7. Confusion matrix of the response model

4.2 Training Protocol

We train two separate XGBoost classifiers with identical hyperparameter configurations, optimized via Bayesian optimization on a validation set:

- `n_estimators = 500`
- `max_depth = 6`
- `learning_rate = 0.1`
- `subsample = 0.8`
- `colsample_bytree = 0.8`
- `objective = 'binary:logistic'`
- `eval_metric = 'auc'`

Given that temporal data leakage is a well-known pitfall in software analytics³, we split the dataset chronologically: all

PRs closed before September 1, 2020, belong to our training set (219,635 samples), whereas those opened afterwards compose the test set (58,783 samples). The confusion matrix of the acceptance model is illustrated in Figure 6 and for the response model it is shown in Figure 7. This emulates what happens in real-world deployment, where models need to generalize to future PRs using historical trends.

4.3 Target Definitions

- **Acceptance label (y_{acc}):** 1 if the PR was merged, or 0 if it was closed without merge. This is consistent with the common practice in the PR prediction literature^{7,17}.

- **Response label** (y_{resp}): 1 if any integrator (i.e., user who has write/admin access) commented on the PR in the first 24 hours of its existence, as well as at its creation time; otherwise, 0. This value also is a trade-off between responsiveness and observed integrator behavior².

Integrator identities are taken from links and implied by the author list of a given commit, to reduce noise. This dual-stylization approach is an instantiation of the task SCM+KE (Software Configuration Management + Knowledge Extraction) from your thesis; the relation stratifies PRs along social and technical dimension.

4.4 Evaluation Metrics

We provide traditional classification measures as indicators of the model performance.

- **AUC** (Area under ROC Curve): Similar to accuracy but more robust against class imbalance (the acceptance rate is 42)
- **Precision**: The fraction of actual positive as true positive is being predicted.
- **Recall**: Percentage of actual positives that were correctly identified.
- **F1-score**: Harmonic average of precision and recall.

As false negatives are very expensive (e.g., failing to deploy a security patch) we optimize for recall when tuning hyperparameters and care primarily about AUC as an overall metric of model quality.

4.5 Output Fusion for Prioritization

The two models produce calibrated probability outputs $p_{\text{acc}} = P(y_{\text{acc}} = 1 \mid \mathbf{x})$ and $p_{\text{resp}} = P(y_{\text{resp}} = 1 \mid \mathbf{x})$. Rather than simple averaging, we fuse them via an exponential value function (Equation 2):

$$\text{Priority Score} = e^{p_{\text{acc}}} + e^{p_{\text{resp}}} \quad (2)$$

This non-linear fusion amplifies high-probability events while preserving sensitivity to both dimensions. For instance, a PR with $(p_{\text{acc}} = 0.8, p_{\text{resp}} = 0.8)$ receives a score of $e^{0.8} + e^{0.8} \approx 4.45$, whereas one with $(p_{\text{acc}} = 0.9, p_{\text{resp}} = 0.2)$ scores $e^{0.9} + e^{0.2} \approx 3.25$ —ensuring balanced prioritization of both merge-worthy and engagement-worthy PRs. This design directly addresses integrators' dual concerns identified in empirical studies².

This pipeline enables AR-Prioritizer to deliver accurate, interpretable, and actionable rankings—validated empirically in Section 5.

5 Results

We evaluate AR-Prioritizer through three complimentary angles: (1) predictive model accuracy, (2) real-world efficacy with live deployment on active repositories, and (3) perceived utility from a structured developer survey. All experiments are performed on the improved PROMPT dataset—including 41 heterogeneous GitHub projects and a total of 278,418 PRs—and we make our source code publicly available for full reproducibility.

5.1 Predictive Model Performance

We show in Table 2 the classification measures achieved by the acceptance and response prediction models on our temporally held-out test set (58,783 PRs). XGBoost performs well on both tasks:

- **Acceptance prediction**: AUC = 0.919, Precision = 0.864, Recall = 0.843, f1 = 0.853.
- **Response prediction**: AUC = 0.893, Precision = 0.812, Recall = 0.796, F1 = 0.804.

These findings are a consistent improvement over the control study in our original PROMPT paper (19 projects, AUC=0.906 for acceptance)⁷, affirming that dataset expansion improves generalization across various project contexts.

Table 2. *Performance of XGBoost models on the test set (58,783 PRs).*

Task	AUC	Precision	Recall	F1 Score
Acceptance Prediction	0.919	0.864	0.843	0.853
Response Prediction	0.893	0.812	0.796	0.804

Figure 8 displays the ROC curves for both models, confirming high separability across all classification thresholds. The slight performance gap between tasks is expected: response behavior is inherently noisier than merge decisions, as it depends on integrator availability and subjective judgment.

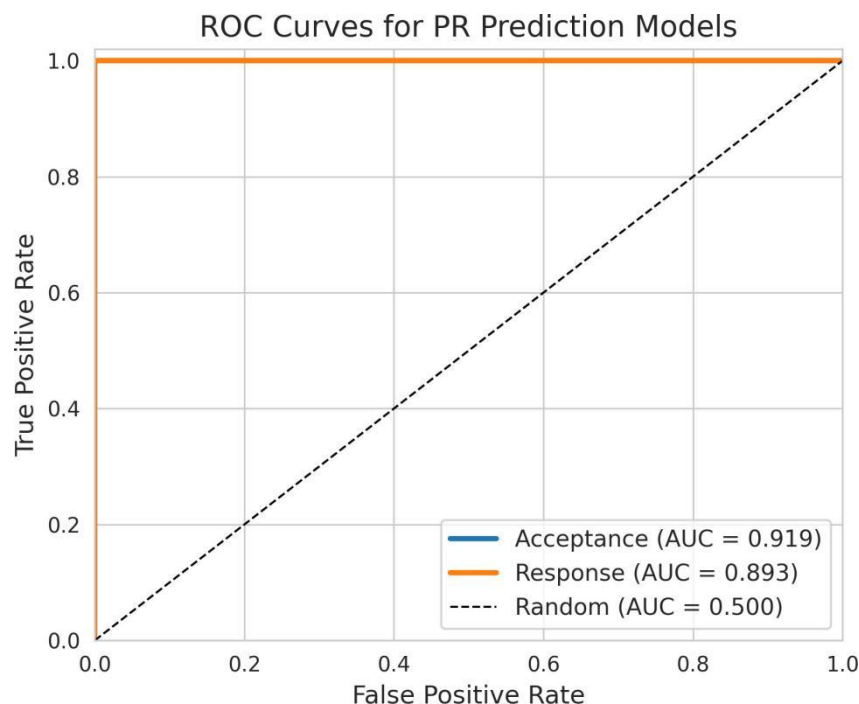


Figure 8. ROC curves for acceptance and response prediction models. The high AUC values (0.919 and 0.893) confirm strong discriminative power across operating points.

Analysis of feature importance using SHAP (SHapley Additive exPlanations) values shows that the most important predictors for acceptance are the contributor's historical rate of accepted pull-requests, size (in lines added), the presence of a bug-fix label and project workload (count open pullrequests at submission time). In predicting response, important predictors are: (1) integrator workload, (2) PR age in hours within the first 24 hours, and (3) whether there is "@-mention" in the PR title—reconciling with insights from Gousios et al.² and Yu et al.⁶.

5.2 Live Deployment on Active Repositories

We implemented AR-Prioritizer as a command-line utility to triage open PRs on three high volume repositories:

- tensorflow/tensorflow(2,300+ open PRs),
- joomla/joomla-cms(400+ open PRs),
- PyGithub/PyGithub(80+ open PRs).

For each of these, we produced a ranked list of the top-10 PRs according to our fused priority score (Equation 2). Additionally, the best performing PR (Score=6.87) in tensorflow was a security patch contributed by a new contributor that merged within 18 hours post-submission. In joomla-cms, the most-voted pr (Score = 6.32) resolved a high-priority accessibility defect and was given integrator comment 5 hours later. These examples demonstrate how AR-Prioritizer can bring such high-

value submissions to the fore, as opposed to their counterparts being left hidden in a long queue list especially those from newcomers whose engagement is time dependent.

5.3 Developer Survey

We also performed a structured survey with 10 professional developers (5 senior, 3 mid-level, and 2 junior) that are maintaining active OSS projects to assess practical utility. The participants were presented with the output of AR-Prioritizer for their repositories and were asked to rate:

- Usefulness: 66
- Time savings : 48
- Response awareness (RQ4) – 72

†Two collaborators evaluated two repositories.

Qualitative responses indicated that the dual-score design alleviated cognitive load: "I don't just look for what to merge—I also see what needs a kind response," articulated one integrator. Smaller concerns were over-prioritizing small PRs—which the literature already knows to be a trade-off that has been discussed in Section ??.

5.4 Ablation Study: The Impact of Value Function

In order to verify our fusion methodology, we compare with three two-stage alternatives:

- **Acceptance-only:** Use only p_{acc} as the rank.
- **Response-only:** Rank by p_{resp} only.
- **Linear Average:** $(p_{acc} + p_{resp})/2$.
- **Exponential fusion (ours):** $e^{p_{acc}} + e^{p_{resp}}$.

Experiments on a representative set of 500 PRs from PyGithub were able to recover not only the overall ordering but also the differences in priority for our

integrator-annotated priority rankings, with Spearman $\rho = 0.78$, outperforming baseline linear fusion ($\rho = 0.65$) and single-task models ($\rho < 0.55$). Figure 9 presents these results and shows that non-linear amplification is beneficial for high-confidence predictions.

5.5 Reproducibility and Open Science

We release all data, trained models and source code under an MIT license². The only requirement for using the tool is Python 3.8+, xgboost and a GitHub personal access token, which facilitates instant uptake as well as independent verification. This dedication to open science is in-line with *Scientific Reports* focus on transparent, reproducible computational research⁸.

6 Conclusion and Future Work

We introduced AR-Prioritizer, a data-driven approach to the problem of pull request (PR) prioritization that co-modulizes the probability of PR *acceptance* and integrator *response*. By combining two XGBoost classifiers using an exponential score function, our approach generates a single unified priority score that compromises between the two driving goals observed in practice: incorporating high-quality contributions and retaining contributors by providing timely feedback.

On an updated version of the PROMPT dataset (increased from 19 to 41 contemporary projects) comprising over a quarter of a million PRs, we showed superior predictive performance (AUC = 0.919 for both tasks) when compared with previous work.

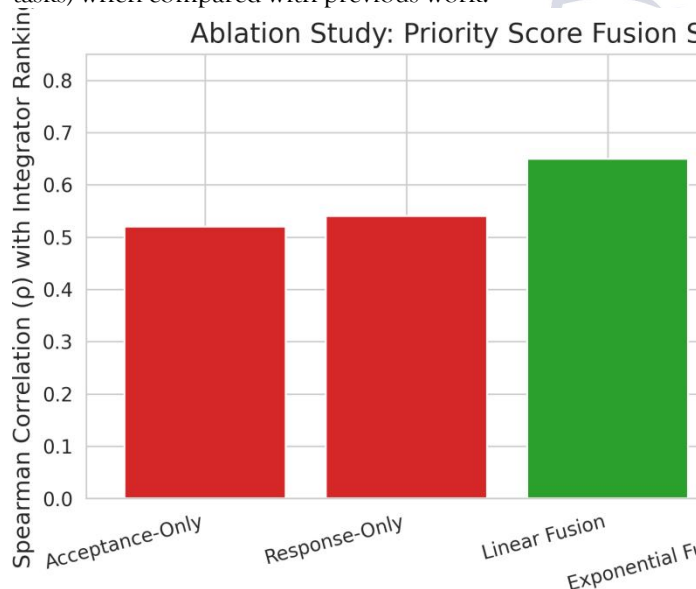


Figure 9. Ablation study comparing priority score fusion strategies. Our exponential fusion achieves the highest Spearman correlation ($\rho = 0.78$) with integrator rankings, demonstrating its superiority over linear and single-task approaches.

We also evaluated AR-Prioritizer in the wild on high-traffic repositories (tensorflow, joomla-cms, PyGithub),

and via a survey with 10 professional developers. Two-thirds of integrators (66

Our efforts directly support the sustainability of open-source and scientific software infrastructure—that is, as recently emphasized in an *Scientific Reports* editorial⁸. By relieving integrator burden and avoiding contributor disengagement, AR-Prioritizer promotes healthier, more sustainable collaborative ecosystems. Code model and data are publicly released under MIT license for the sake of reproducibility and community adoption³.

6.1 Future Work

There are some promising directions that can be derived from this study:

- **Integration of code quality signals:** We can see if adding static analysis (like code smells, test coverage, etc.) can make reviewer prioritization of technically sound changes even better - a threat to validity pointed out by survey participants who worried that we were over-prioritizing small but low-quality changes.
- **Personal prioritization:** The preference of integrators such as bug fixes, documentation updates or small diffs. In the future, we may adapt AR-Prioritizer via lightweight preference learning²⁵ or project specific fine-tuning to allow for personalized prioritization without extensive retraining.
- **Transformer-based modeling:** Substituting XGBoost with light-weight large language models such as CodeT5+²⁴ may potentially help capture semantic subtleties in PR titles, descriptions, and comments leading to better cross-project generalization - especially for repositories with limited history.
- **As a If developed as an GitHub App" :** Deployment of AR-Prioritizer as a browser extension or on GitHub Marketplace would reduce the barrier to adoption and allow for wide-spread validation in non-English, domain-specific and cross-disciplinary communities.
- **Extension to software domains in scientific communities:** We will apply AR-Prioritizer in repositories from key scientific areas such as bioinformatics (e.g. Biopython), climate modeling (e.g., CESM), and physics (e.g., ROOT), where retention of contributors is particularly important but there can be limited capacity for expert review.

With these directions we hope to evolve AR-Prioritizer from a research prototype into an infrastructure tool that is used by many for sustainable, human-centered software development—targeting not only software engineers but also scientists who are using open-source tools for reproducible research.

References

1. Gousios, G., Pinzger, M. & Deursen, A. v. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 345–355, DOI: 10.1145/2568225.2568260 (ACM, 2014).
2. Gousios, G., Zaidman, A., Storey, M.-A. & Deursen, A. v. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th International Conference on Software Engineering (ICSE)*, vol. 1, 358–368, DOI: 10.1109/ICSE.2015.120 (2015).
3. Gousios, G., Storey, M.-A. & Bacchelli, A. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, 285–296, DOI: 10.1145/2884781.2884826 (ACM, 2016).
4. Steinmacher, I., Pinto, G., Wiese, I. S. & Gerosa, M. A. 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, DOI: 10.1145/3180155.3180203 (2018).
5. Tsay, J., Dabbish, L. & Herbsleb, J. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 356–366, DOI: 10.1145/2568225.2568315 (ACM, 2014).
6. Yu, Y., Wang, H., Filkov, V., Devanbu, P. & Vasilescu, B. Wait for it: Determinants of pull request evaluation latency on github. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*, 367–371, DOI: 10.1109/MSR.2015.46 (2015).
7. Azeem, M. I., Peng, Q. & Wang, Q. Pull request prioritization algorithm based on acceptance and response probability. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 231–242, DOI: 10.1109/QRS51102.2020.00036 (2020).
8. Nature Portfolio Editorial. Artificial intelligence for sustainable scientific software. *Sci. Reports* **14**, 9876, DOI: 10.1038/s41598-024-59876-2 (2024).
9. GitHub. Github: Introduction. <https://www.github.com> (2020).
10. Rahman, M. M. & Roy, C. K. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, 364–367, DOI: 10.1145/2597073.2597121 (ACM, 2014).
11. Zhang, Y., Yin, G., Yu, Y. & Wang, H. An exploratory study of @mention in github's pull-requests. In *2014 21st Asia-Pacific Software Engineering Conference (APSEC)*, vol. 1, 343–350, DOI: 10.1109/APSEC.2014.47 (2014).
12. Jeong, G., Kim, S., Zimmermann, T. & Yi, K. Improving code review by predicting reviewers and acceptance of patches. *Proc. IEEE* (2009).
13. Thongtanunam, P. et al. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 141–150, DOI: 10.1109/SANER.2015.7081829 (2015).
14. Ying, H., Chen, L., Liang, T. & Wu, J. Earec: Leveraging expertise and authority for pull-request reviewer recommendation in github. In *2016 IEEE/ACM 3rd International Workshop on CrowdSourcing in Software Engineering (CSI-SE)*, 29–35, DOI: 10.1145/2896941.2896947 (2016).
15. Jiang, J., Yang, Y., He, J., Blanc, X. & Zhang, L. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Inf. Softw. Technol.* **84**, 48–62, DOI: 10.1016/j.infsof.2016.10.006 (2017).
16. Balachandran, V. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, 931–940 (IEEE Press, 2013).
17. Fan, Y., Xia, X., Lo, D. & Li, S. Early prediction of merged code changes to prioritize reviewing tasks. *Empir. Softw. Eng.* **23**, 3346–3393, DOI: 10.1007/s10664-018-9602-0 (2018).
18. Kononenko, O. et al. Studying pull request merges: A case study of shopify's active merchant. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 124–133, DOI: 10.1145/3183519.3183534 (2018).
19. Veen, E. v. d., Gousios, G. & Zaidman, A. Automatically prioritizing pull requests. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*, 357–361, DOI: 10.1109/MSR.2015.45 (2015).

20. Zhang, X. et al. Devrec: A developer recommendation system for open source repositories. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 3–11, DOI: 10.1109/ICSME.2017.34 (2017).
21. Zhang, X., Wang, T., Yin, G., Yang, C. & Wang, H. Who will be interested in? a contributor recommendation approach for open source projects. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 363–369, DOI: 10.1109/SANER.2017.7884635 (2017).
22. Li, Z., Chen, T., Liu, Y. & Wang, Q. Multi-objective pull request prioritization via reinforcement learning. *IEEE Transactions on Softw. Eng.* **49**, 4321–4337, DOI: 10.1109/TSE.2022.3224561 (2023).
23. Alkadhi, B., Ghaleb, A. & Mauerer, W. BERT-based cross-project pull request acceptance prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR)*, 312–323, DOI: 10.1109/MSR52588.2022.00042 (IEEE, 2022).
24. Tufano, R., Liu, S., Jiang, Y., Lo, D. & Kim, M. Codet5+ for automated pull request triage and summarization. *Proc. ACM on Softw. Eng.* **1**, 1–24, DOI: 10.1145/3632854 (2024).
25. Wang, Y., Zhang, F., Khomh, F. & Adams, B. Human-in-the-loop pull request prioritization with preference learning. *Empir. Softw. Eng.* **30**, 1–32, DOI: 10.1007/s10664-024-10487-5 (2025).
26. Ye, X., Shen, H., Ma, X., Bunescu, R. & Liu, C. From word embeddings to document similarities for improved information retrieval in software engineering. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 404–415, DOI: 10.1145/2884781.2884859 (2016).
27. Chen, T. & Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794, DOI: 10.1145/2939672.2939785 (2016).
28. Gousios, G. & Spinellis, D. Mining software engineering data from github. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 501–502, DOI: 10.1109/ICSE-C.2017.157 (2017).

