Exploring the Impact of GitHub Actions on Pull Request Reviews in Machine Learning Projects

João Helis Bernardo
Federal University of Rio Grande do Norte
Federal Institute of Rio Grande do Norte
Natal, Brazil
joao.helis@ifrn.edu.br

Daniel Alencar da Costa
University of Otago
Dunedin, New Zealand
danielcalencar@otago.ac.nz

Sérgio Queiroz de Medeiros
Federal University of Rio Grande do Norte
Natal, Brazil
sergio.medeiros@ufrn.br

Uirá Kulesza
Federal University of Rio Grande do Norte
Natal, Brazil
uira@dimap.ufrn.br

ABSTRACT

Continuous Integration (CI) tools like GitHub Actions were originally designed to streamline development workflows in traditional software systems by automating tasks such as building and testing, which has proven beneficial in improving review efficiency. However, ML projects present additional complexities—such as non-determinism, challenging testing processes, and longer build durations—that may limit the effectiveness of CI in supporting efficient reviews. Given these unique challenges, it is essential to reassess how CI tools impact the review process within ML contexts. This study empirically investigates the impact of GitHub Actions on PR review dynamics across 55 GitHub-based ML projects, focusing on metrics such as time to close a PR (i.e., PR latency), PR churn, comments, and PR submission frequency. Using a Regression Discontinuity Design (RDD), we analyze PR data from 12 months *before* and *after* the adoption of GitHub Actions. Our results show that GitHub Actions does not significantly reduce PR review times in ML projects, with factors such as PR churn and backlog size playing a larger role in influencing review efficiency. Additionally, rejected PRs were characterized by higher churn and more extensive discussions. These findings suggest that, while CI tools automate repetitive tasks and reduce manual workload, they may not fully address the unique demands of ML project reviews. We provide practical recommendations to enhance review efficiency in ML workflows, including strategies for incremental PR submissions and optimized backlog management.

KEYWORDS

continuous integration, machine learning, github actions, software engineering for ML, review efficiency

1 Introduction

Machine Learning (ML) projects are software projects that incorporate ML components to enable more intelligent functionalities. These projects typically fall into two broad categories: (i) *ML tools*, such as libraries and frameworks that offer reusable ML capabilities (e.g., `scikit-learn`); and (ii) *ML-enabled applications*, which integrate ML models to support domain-specific tasks (e.g., image recognition or recommendation systems). In this study, we adopt a broad definition of ML projects that includes both categories.

Developing ML projects requires the combined effort of mathematical modeling and software engineering [28]. Beyond implementing functionality, ML developers must manage data dependencies, evaluate model performance, and handle non-deterministic behaviors [27]. These complexities make ML projects inherently more difficult to manage than traditional software systems and may introduce additional difficulties in collaborative workflows such as code review.

Open-source software (OSS) development often follows a pull-based model, where contributors propose changes via pull requests (PRs), and integrators review them before merging [9, 13]. This model fosters collaboration but also introduces overhead—especially in ML projects, where PR reviews must consider not only code correctness but also data, configuration, and model performance implications [1, 8, 17, 29]. The probabilistic behavior of ML models further complicates validation and reproducibility during review [1].

To reduce manual effort and streamline review workflows, OSS projects often adopt CI tools that automate tasks such as building, testing, and deploying code. GitHub Actions, a GitHub's native CI solution, integrates these tasks into the pull-based workflow [31]. In traditional software projects, CI has been shown to improve code quality and reduce PR review delays [2, 3, 10, 11, 14, 22, 23, 26, 31, 33]. However, it remains unclear whether these benefits translate to ML projects, where review complexity often stems from factors not easily addressed by conventional CI workflows.

This study aims to empirically assess the impact of GitHub Actions on PR review efficiency in ML projects. We analyze data from 55 open-source ML projects on GitHub to evaluate changes in PR latency (i.e., time to close a PR) before and after the adoption of this CI service. Additionally, we examine the characteristics of merged and rejected PRs to identify factors that influence review outcomes. By doing so, we contribute empirical evidence on how CI services may affect the development dynamics of ML teams and offer insights into the limitations of current automation practices.

Our investigation is guided by the following research questions (RQs):

- ***RQ1. What is the impact of GitHub Actions on the time to review pull requests in ML projects?***
  We examine whether the adoption of GitHub Actions impacts the time to close PRs (i.e., PR latency) in ML projects.

Our results indicate that, although the adoption of GitHub Actions did not significantly affect PR latency, higher PR submission rates and active project engagement were associated with shorter review times. In contrast, backlog accumulation, complex code changes, and extensive discussions contributed to longer PR latency.

- ***RQ2. How do the characteristics of merged and rejected pull requests differ in ML projects?***
  We analyze the characteristics of merged versus rejected PRs. In ML projects, rejected PRs tend to exhibit higher churn, longer latency, and a greater number of comments, suggesting that complex and discussion-heavy contributions are less likely to be accepted. To improve their chances of acceptance, contributors should aim to submit manageable PRs with clear and concise changes.

Paper organization. Section 2 discusses relevant prior work. Section 3 describes our study design. Section 4 presents the findings, followed by a discussion in Section 5. Section 6 addresses threats to validity, and Section 7 concludes the paper. Finally, Section 8 outlines directions for future work.

2 Related Work

Prior studies have shown that CI can improve PR review efficiency in traditional open-source software projects. For example, Cassee et al. [5] report that the adoption of Travis CI reduces PR discussion by up to one comment per review.

In contrast, fewer studies have investigated the effectiveness of CI in ML projects, which present unique challenges such as model validation, data dependency management, and non-deterministic behavior. Rzig et al. [21] conducted a large-scale study characterizing the use of CI tools in ML projects. They found that CI is commonly used to support automation and quality control but often fails to reduce review time, largely due to the unique demands of ML workflows. While their work sheds light on how CI is used in ML projects, it does not evaluate the impact of CI on collaborative development practices such as PR reviews.

Bernardo et al. [4] performed a large-scale empirical study comparing CI practices between ML and non-ML projects on GitHub. Their findings indicate that ML projects tend to have longer build durations and lower test coverage. Through quantitative and qualitative analyzes, they highlighted challenges specific to ML contexts, such as mistrust in CI outcomes and greater complexity in maintaining reproducible workflows. While their study focuses on characterizing the presence and challenges of CI usage, our work extends this line of inquiry by examining how the adoption of CI services—specifically *GitHub Actions*—affects PR review dynamics in ML projects.

Our work is also informed by Wessel et al. [31], who applied Regression Discontinuity Design (RDD) to evaluate the impact of GitHub Actions adoption in OSS projects. They reported a significant reduction in PR latency post-adoption and found that increased discussion correlates with longer review times—a pattern consistent with our findings. However, their study did not include ML projects. We build on their methodology by applying RDD specifically to ML repositories, offering new insights into how CI adoption affects PR review dynamics in this domain. To the best of our knowledge, no prior work has empirically assessed the impact of CI services—specifically GitHub Actions—on PR review efficiency in ML projects.

3 Research Methodology

In this section, we explain how we select the studied projects and construct the database that we use in our analyses.

3.1 Studied Projects

To investigate the impact of GitHub Actions on the PR review dynamics of ML projects, we selected a dataset of open-source ML projects from GitHub that consistently used GitHub Actions *CI workflows*. *CI workflows* are pipelines configured to perform CI-related tasks, such as executing automated tests. We started by selecting the dataset curated by Bernardo et al. [4], which includes 93 ML projects and 92 non-ML projects with an active history of GitHub Actions *CI workflows* usage. For our analysis, we focused exclusively on the ML projects. The ML projects in our dataset encompass a range of ML frameworks and libraries, such as *scikit-learn*, as well as ML applications like *Faceswap*. These projects leverage ML techniques or components to fulfill specific user needs or provide general-purpose functionalities.

Following similar methodologies used in prior studies on CI tools adoption in open-source projects [5, 23, 30, 31, 33], we analyzed a 24-month period surrounding the adoption of GitHub Actions *CI workflows* for each project, consisting of 12 months *before* and 12 months *after* the adoption. The adoption date was determined by the first recorded execution (*Run*) of a GitHub Actions *CI workflow* in the project's history. To ensure we selected projects with consistent activity, we filtered projects with at least one PR submitted each month throughout the 24-month analysis period. This step was used to avoid periods of project inactivity and ensure a robust analysis of PR review dynamics. As a result, 55 ML projects remain in our analysis.

3.2 Data Collection

After selecting our studied projects, we began collecting project metadata. Each step of the process is detailed below.

Step 1: Collection of projects metadata. We use the GitHub API to collect general information for each studied project, such as primary language and number of stars, as well as the entire PR and commit history of the projects within their default branch. In Table 1, we present the general descriptive statistics for the investigated projects, including the number of PRs per project, the age of the projects in years, and the duration in years that each project has been using GitHub Actions.

Step 2: Compute Metrics. We use data from Step 1 to compute the metrics for our analyses. We segment the project history into 30-day periods (one month), as illustrated in Figure 1. We exclude data from the 30 days surrounding the adoption of GitHub Actions *CI workflows*, as this period is well-documented in the literature as an adaptation phase for CI services [5, 30, 31, 33]. The variables used in this study are based on metrics established in previous literature [31, 33], and were collected using the GitHub API on May 2024. These variables include measures of project activity, collaboration,

Table 1: Descriptive statistics for the investigated projects.

Metric	Min	Q1	Median	Mean	Q3	Max
Pull Requests	290	985	2023	3452	3516	26908
Commits	698	2592	5509	7387	9218	31527
Integrators	5	29.5	52	105.6	106.5	714
Contributors	13	104	202	356.9	332	3159
Proj. Age (y)	5.1	5.9	7	7.9	9.6	14.8
GHA usage (y)	1.9	3.2	3.2	3.2	3.5	3.7

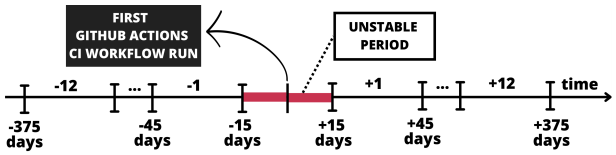

Figure 1: Periods of Analysis.

and PR characteristics. A detailed description of each variable is provided in Table 2.

Step 3: Perform Analysis. The curated datasets produced in Step 2 are the basis for the subsequent analyses for RQ1 and RQ2. All the scripts and datasets we used in our analyses are available in our online Appendix.[1]

4 Results

In this section, we present the motivation, approach, and results for each RQ.

RQ1. What is the impact of GitHub Actions on the time to review pull requests in ML projects?

Motivation. Efficient PR reviews are crucial for maintaining development speed and code quality in ML projects, where complex dependencies and computational tasks might slow the process. While GitHub Actions automates tasks to streamline builds and tests, its effectiveness in reducing PR review time for ML projects remains uncertain. Assessing the impact of GitHub Actions on PR latency helps determine the extent to which CI automation can address ML-specific review challenges.

Approach. To address *RQ*1, we employed a Regression Discontinuity Design (RDD) to specifically analyze the effect of GitHub Actions on PR latency, following the work of Zhao et al. [33] and Wessel et al. [31]. RDD is a quasi-experimental technique that allows for the analysis of an intervention (in this case, the adoption of GitHub Actions) by examining changes in outcome variables at the intervention point while controlling for confounding factors [15, 16, 25]. Using RDD, we model our variables of interest as a function of time, looking for discontinuities around the intervention point [5]. If the intervention does not affect the outcome, there will be no discontinuity, and the outcome will remain continuous over time [7]. The model behind RDD is as follows:

$$y_i = \alpha + \beta \cdot \text{time}_i + \gamma \cdot \text{intervention}_i + \delta \cdot \text{time\_after\_intervention}_i + \eta \cdot \text{controls}_i + \epsilon_i,$$

where i represents the observations for each project.

To account for the evolution of our variables over time and the adoption of GitHub Actions, we use three variables: *time*, *time after intervention*, and *intervention*. The *time* variable is recorded in months from the start to the end of the observation period for each project. The *intervention* variable is a binary indicating whether the time point j is before (*intervention* = 0) or after (*intervention* = 1) the adoption event. The *time after intervention* variable measures the number of months since the adoption of GitHub Actions at time point j, and is set to 0 before adoption. The controls$_i$ variables help isolate the effects of GitHub Actions adoption from other factors that could influence the dependent variables. For observations before the intervention, with controls held constant, the regression line slope is β, and after the intervention, the slope becomes $\beta+\delta$. The magnitude of the intervention effect is represented by the difference γ between the two regression values of y_i at the intervention point.

We used mixed-effects linear regression using the *lmerTest* R package [18], with project name and primary programming language modeled as random effects to account for project-to-project and language-to-language variability [12]. All other variables were treated as fixed effects. Variables with high variance were log-transformed [24], and we addressed multicollinearity by removing variables with a Variance Inflation Factor (VIF) greater than 5 [24].

The model's goodness of fit was assessed using the marginal R^2 (R^2_m), indicating the variance explained by the fixed effects, and the conditional R^2 (R^2_c), which captures the variance explained by both fixed and random effects [19]. We reported statistically significant coefficients ($p < 0.05$) and estimated variance using ANOVA.

Results. The impact of GitHub Actions on PR latency was assessed using a mixed-effects RDD model with PR latency as the dependent variable. Figure 2 shows trends in PR latency *before* and *after* the adoption of GitHub Actions, while Table 3 provides coefficients and significance levels for model variables. The model's marginal R^2_m is 0.47, indicating that 47% of the variance in PR latency is explained by the fixed effects, and the conditional R^2_c of 0.65 accounts for variability across projects and languages. These results suggest that while fixed effects capture significant variability, project-specific differences also play a crucial role in PR latency.

The analysis reveals that the number of submitted PRs had a strong negative effect on PR latency ($\beta = -0.507$, $p < 0.001$), with the highest sum of squares (137.9). This suggests that projects with higher PR submission rates experience shorter latency, likely due to increased project activity fostering quicker decision-making. Conversely, a higher number of open PRs at the beginning of each period correlated with longer latency ($\beta = 0.3996$, $p < 0.001$), highlighting a backlog effect where accumulated reviews slow down the reviewing efficiency.

PR characteristics also significantly affected the PR latency of ML projects. PRs with higher churn ($\beta = 0.0829$, $p < 0.01$) and more commits ($\beta = 0.2653$, $p < 0.001$) exhibited longer latency, likely due to the need for more comprehensive review evaluations. Interestingly, PRs with more changed files showed shorter latency

[1] https://zenodo.org/records/14050696

Table 2: Descriptions and rationale for variables used in the analysis.

Variable	Variable Explanation (D: Description | R: Rationale)
priorTravisCIUsage	**D:** Binary indicator of prior experience with *Travis CI* before adopting GitHub Actions. **R:** Accounts for the impact of prior CI experience on PR processing efficiency.
projectIntegrators	**D:** The number of integrators involved in reviewing and PRs. **R:** Captures collaboration level and its influence on PR processing.
previousOpenPRs	**D:** The number of open PRs at the start of each observation period. **R:** Measures the impact of backlog on PR processing times.
submittedPRs	**D:** The number of PRs submitted during each observation period. **R:** Reflects the level of project activity and momentum affecting PR processing efficiency.
PRsChurn	**D:** Median churn (sum of lines added and removed) in closed PRs for each period. **R:** Measures the complexity of PR changes and its effect on PR processing time.
PRsCommits	**D:** Median number of commits in closed PRs for each period. **R:** Accounts for the additional effort needed to review PRs with multiple commits.
PRsParticipants	**D:** Median number of participants in closed PRs for each period. **R:** Reflects the level of collaboration or scrutiny in each PR review process.
PRsChangedFiles	**D:** Median number of changed files in closed PRs for each period. **R:** Measures the size and structure of PRs and their impact on review times.
PRsComments	**D:** Median number of comments in closed PRs during each period. **R:** Serves as a proxy for the intensity of discussions and feedback in PR reviews.
PRLatency	**D:** The time taken to close a PR, measured from the time it was opened to when it was merged or rejected. **R:** Serves as the primary indicator of PR processing efficiency and the dependent variable in this study.

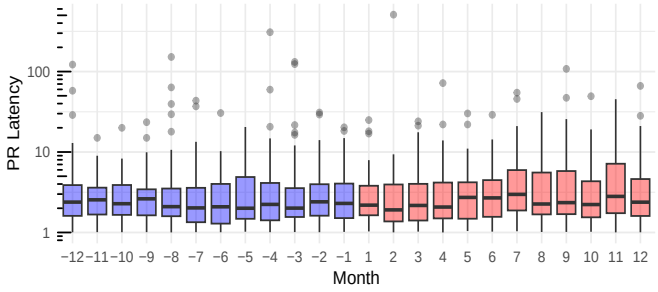

Figure 2: Trends in the PR Latency of ML projects *before* (blue) and *after* (red) GitHub Actions adoption.

($\beta = -0.2095$, $p < 0.001$), possibly indicating that these PRs were better structured or involved modular changes that are easier to review.

Additionally, the number of PR comments was associated with longer latency ($\beta = 0.3671$, $p < 0.001$), suggesting that extensive discussions tend to extend review time. Projects with prior experience using Travis CI were associated with shorter latency ($\beta = -0.4261$, $p < 0.05$), indicating that familiarity with CI practices enhances review efficiency, though the relative impact was modest.

The model's time-based predictors (*time*, *intervention*, and *time_after_intervention*) for GitHub Actions adoption were not statistically significant, indicating no immediate or abrupt change in PR latency following GitHub Actions adoption. As shown in Figure 2, the PR latency trend remained stable *before* and *after* GitHub Actions adoption, with a slight increase post-adoption, potentially reflecting initial workflow adjustments to new CI processes.

	Coeffs	Sum Sq
(Intercept)	1.6419	
priorTravisCIUsageTRUE	-0.4789*	0.143
log(projectIntegrators)	-0.0645	0.705
log(previousOpenPRs)	0.3996***	35.407
log(submitedPRs)	-0.507***	137.944
log(PRChurn)	0.0829**	7.794
log(PRCommits)	0.2653***	7.116
log(PRParticipants)	0.0761	2.603
log(PRChangedFiles)	-0.2095***	6.467
log(PRComments)	0.3671***	19.977
time	-0.0066	0.007
interventionTRUE	-0.004	0.001
time_after_intervention	0.013	0.659
R^2_m	0.47	
R^2_c	0.65	

*** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$

Table 3: The effects of GitHub Actions on Pull Request Latency. The response variable is *log*(PRLatency).

While GitHub Actions adoption did not significantly impact PR latency, higher PR submissions and active project engagement correlated with shorter review times, whereas backlog accumulation, complex changes, and extensive discussions contributed to longer PR latency.

RQ2. How do the characteristics of merged and rejected pull requests differ in ML projects?

Motivation. Understanding the characteristics that distinguish merged from rejected PRs in ML projects can provide insights into factors influencing PR acceptance. Identifying these characteristics allows contributors to better tailor their submissions and maintainers to refine their review criteria, ultimately enhancing the efficiency and quality of the review process in ML workflows.

Approach. To analyze the differences in PR characteristics between merged and rejected PRs, we employed the Mann-Whitney-Wilcoxon (MWW)[32] test to identify statistically significant differences in key variables, such as churn, commits, and comments. Additionally, we calculated Cliff's Delta[6] to measure the effect size of these differences, indicating their practical significance. We applied the thresholds suggested by Romano et al. [20], interpreting the effect sizes as follows: $\delta < 0.147$ (negligible), $\delta < 0.33$ (small), $\delta < 0.474$ (medium), and $\delta \geq 0.474$ (large).

Results. When examining the characteristics of contributed code in ML projects, we found no statistically significant differences between merged and rejected PRs in terms of the number of commits or changed files. Both types of PRs had a median of 2 commits, with a negligible effect size (p = 0.755), and a median of 2 changed files, also with a negligible effect size (p = 0.371). This indicates that the number of commits and the scope of affected files are not key factors influencing the decision to merge or reject a PR.

However, we identified a notable difference in PR churn, where rejected PRs had a higher median churn (48.25) compared to merged PRs (36.75). This difference, although marginally significant (p = 0.058) with a small effect size, suggests that PRs involving more extensive modifications are more likely to be rejected. Higher churn could introduce complexity and increase the risk of errors, prompting maintainers to exercise caution and be more critical of such PRs.

We also examined communication efforts, particularly PR comments and participants. Rejected PRs had a significantly higher number of comments (median = 2) compared to merged PRs (median = 1), with a medium effect size ($p = 1.930e-04$). This indicates that rejected PRs often involve more extensive discussions, potentially due to unresolved issues or the need for multiple rounds of feedback and revisions. While rejected PRs tended to involve more participants in their discussions, this difference was not statistically significant ($p = 0.088$), suggesting that the number of participants has a limited impact on PR outcomes.

The most pronounced difference between merged and rejected PRs was in PR latency. Merged PRs had a median latency of 1.05 days, whereas rejected PRs had a significantly longer median latency of 6.16 days, with a large effect size ($p < 0.001$). This finding highlights that rejected PRs experience protracted review cycles, likely due to the need for additional revisions and feedback iterations.

> Rejected PRs in ML projects are characterized by higher churn, longer latency, and more comments, indicating that complex and discussion-heavy contributions are less likely to be merged. Contributors should strive to submit manageable PRs with clear, concise changes to improve their chances of acceptance.

5 Discussion and Implications

In the following, we discuss the results and their implications for practice and research.

Tailoring CI for the unique demands of ML projects. The contrasting impact of GitHub Actions adoption between traditional OSS and ML projects highlights an opportunity to refine CI configurations specifically for ML workflows. Different from our results, Wessel et al. [31] observed that GitHub Actions effectively reduces PR latency in traditional OSS, which might indicate that the complexities of ML projects, such as model validation, data handling, and iterative testing, might require tailored CI tools that extend beyond standard automation capabilities. Traditional CI setups, optimized for code-centric tasks, may not fully support the model-focused, data-intensive nature of ML development. This gap indicates a need for CI advancements that can automate ML-specific tasks, such as model performance verification, dataset versioning, and dependency checks. By addressing these ML-specific requirements, CI tools could help alleviate the extended review times often associated with ML projects and improve the overall efficiency of PR processes. These findings emphasize the importance of developing CI solutions tailored to the ML domain, focusing on reducing data bottlenecks and model management within CI pipelines.

Beyond Review Speed: The Hidden Benefits of CI in ML Projects. While GitHub Actions did not significantly reduce PR latency, its seamless integration into ML project pipelines could offer other valuable benefits. ML projects that adopt CI use it for building, testing, and code analysis, which is crucial for maintaining quality [21]. By automating essential tasks of the project development, GitHub Actions potentially alleviates the cognitive load on reviewers, allowing them to focus on higher-level aspects of the PR. This could improve code quality and project stability, even if review times are not significantly reduced. Additionally, projects with established CI practices (e.g., previous use of Travis CI) exhibited shorter PR latencies, suggesting that the cumulative experience with CI tools and workflows contributes to more efficient reviews over time.

Practical Recommendations for Contributors and Maintainers. Our findings for RQ1 showed that an increased number of open PRs correlates with longer review times. Furthermore, our findings for RQ2 highlight the nuanced relationship between PR complexity, communication, and review outcomes. Rejected PRs were characterized by higher churn and more extensive discussions, indicating that substantial changes often lead to more prolonged and iterative feedback cycles. This suggests that contributors should carefully consider the scope of their changes and aim to submit smaller, more manageable PRs to minimize complexity and reduce the likelihood of rejection. Based on these findings, we propose

several actionable recommendations for ML project contributors and maintainers:

- ***Focus on Backlog Management:*** Since a higher number of open PRs was linked to longer review times, maintainers should establish regular review cycles to manage the backlog effectively and avoid processing delays.
- ***Minimize PR Complexity:*** Contributors should aim to submit incremental changes to avoid high churn and reduce the likelihood of rejection. High churn was strongly associated with longer PR latency and higher rejection rates.
- ***Prioritize Early Feedback:*** Effective communication and early issue identification are crucial. Implementing automated feedback mechanisms and establishing clear review guidelines can help reduce the number of comments and iterations needed during reviews.

6 Threats to Validity

This study has limitations that may affect the generalizability and interpretation of its findings.

Construct Validity. We did not analyze the specific content or purpose of the GitHub Actions workflows. As such, we cannot determine whether CI was used to automate ML-specific tasks—such as model training, data validation, or inference—or limited to conventional software tasks like linting or unit testing. This limits our ability to interpret why CI adoption had no significant effect on PR latency. Additionally, we measured review dynamics using observable metrics such as latency, churn, and comment count, which may not fully capture more nuanced factors like model correctness, data complexity, or the nature of reviewer feedback.

Internal Validity. Our study is based on observational data, which makes causal inference inherently difficult. While we applied a Regression Discontinuity Design (RDD) to minimize confounding, unobserved variables—such as contributor expertise, code ownership, or simultaneous process changes—could still influence the observed outcomes. We also accounted for prior CI usage via Travis CI, but did not include other tools like Jenkins, which may also shape a project's CI maturity. Nevertheless, previous studies [14] have identified Travis CI as the most widely adopted CI service on GitHub prior to the introduction of GitHub Actions.

External Validity. Our dataset includes 55 GitHub-based ML projects, which may not capture the full diversity of ML software development across different domains, organizational contexts, or CI adoption patterns. Notably, our sample contains both ML libraries and ML applications, which may follow distinct development workflows. Aggregating them may have introduced variability that affects the consistency of our results. Moreover, we only studied projects using GitHub Actions, and the findings may not generalize to projects using other CI tools such as Travis CI or GitLab CI.

7 Conclusion

This study investigated the impact of GitHub Actions on PR review dynamics in ML projects by analyzing data from 55 GitHub-based repositories. Contrary to expectations drawn from prior research on traditional software projects, our results show that the adoption of GitHub Actions did not significantly reduce PR review latency in ML contexts. Instead, we observed that intrinsic PR characteristics—such as churn and the number of comments—had a stronger association with review outcomes. Rejected PRs tended to exhibit higher churn, more extensive discussions, and longer review cycles, suggesting that complexity and communication demands play a central role in PR evaluation.

These findings indicate that while CI tools like GitHub Actions provide valuable automation capabilities, their current usage in ML projects may not sufficiently address factors that influence review efficiency. Further investigation is needed to understand how CI tools is applied within ML workflows and to identify opportunities for better alignment between automation and the specific challenges of ML development.

8 Future Work

Our findings highlight several avenues for future research:

- **CI Workflow Analysis in ML Contexts:** Future studies should examine the actual content of GitHub Actions workflows to determine whether they automate ML-specific tasks—such as model training, inference testing, or data validation—or focus only on general software tasks. Understanding this scope is essential to interpret the limited impact observed.
- **Project Type Stratification:** Since our dataset includes both ML libraries and ML applications, future work could disaggregate these categories to investigate whether the impacts of the adoption of CI services such as GitHub Actions differ by project type.
- **Qualitative PR Analysis:** A complementary qualitative analysis of merged and rejected PRs—particularly those with high churn and extensive discussions—could uncover review dynamics not captured by quantitative metrics. Additionally, our results could be further enriched through surveys and interviews with developers and project leaders of ML projects, offering deeper insights into the specific challenges, expectations, and contextual factors that influence PR evaluation in this domain.

ARTIFACT AVAILABILITY

For replication purposes, we publicize our datasets and scripts to the interested researchers: https://zenodo.org/records/14050696.

ACKNOWLEDGMENTS

This work is partially supported by INES.IA (www.ines.org.br), CNPq grant 408817/2024-0.

REFERENCES

[1] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. 2018. Software engineering challenges of deep learning. In *2018 44th euromicro conference on software engineering and advanced applications (SEAA)*. IEEE, 50–59.

[2] João Helis Bernardo, Daniel Alencar da Costa, and Uirá Kulesza. 2018. Studying the impact of adopting continuous integration on the delivery time of pull requests. In *Proceedings of the 15th International Conference on Mining Software Repositories*. 131–141.

[3] João Helis Bernardo, Daniel Alencar da Costa, Uirá Kulesza, and Christoph Treude. 2023. The impact of a continuous integration service on the delivery time of merged pull requests. *Empirical Software Engineering* 28, 4 (2023), 97. doi:10.1007/s10664-023-10327-6

[4] João Helis Bernardo, Daniel Alencar da Costa, Sergio Queiroz de Medeiros, and Uirá Kulesza. 2024. How do Machine Learning Projects use Continuous Integration Practices? An Empirical Study on GitHub Actions. In *Proceedings of the 21th International Conference on Mining Software Repositories*.

[5] Nathan Cassee, Bogdan Vasilescu, and Alexander Serebrenik. 2020. The silent helper: the impact of continuous integration on code reviews. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 423–434.

[6] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin* 114, 3 (1993), 494.

[7] Thomas D Cook and D T Campbell. 1979. *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin.

[8] Elizamary de Souza Nascimento, Iftekhar Ahmed, Edson Oliveira, Márcio Piedade Palheta, Igor Steinmacher, and Tayana Conte. 2019. Understanding development process of machine learning systems: Challenges and solutions. In *2019 acm/ieee international symposium on empirical software engineering and measurement (esem)*. IEEE, 1–6.

[9] Tapajit Dey and Audris Mockus. 2020. Effect of technical and social factors on pull request quality for the npm ecosystem. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[10] Wagner Felidré, Leonardo Furtado, Daniel A da Costa, Bruno Cartaxo, and Gustavo Pinto. 2019. Continuous integration theater. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–10.

[11] Guilherme Freitas, João Helis Bernardo, Gustavo SizíLio, Daniel Alencar Da Costa, and Uirá Kulesza. 2023. Analyzing the Impact of CI Sub-practices on Continuous Code Quality in Open-Source Projects: An Empirical Study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*. 1–10.

[12] A. Gałecki and T. Burzykowski. 2013. *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*. Springer New York. https://books.google.com.br/books?id=rbk_AAAAQBAJ

[13] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*. 345–355.

[14] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*. 426–437.

[15] Ray Hyman. 1982. Quasi-experimentation: Design and analysis issues for field settings (book). *Journal of Personality Assessment* 46, 1 (1982), 96–97.

[16] Guido W Imbens and Thomas Lemieux. 2008. Regression discontinuity designs: A guide to practice. *Journal of econometrics* 142, 2 (2008), 615–635.

[17] Foutse Khomh, Bram Adams, Jinghui Cheng, Marios Fokaefs, and Giuliano Antoniol. 2018. Software engineering for machine-learning applications: The road ahead. *IEEE Software* 35, 5 (2018), 81–84.

[18] Alexandra Kuznetsova, Per B Brockhoff, and Rune Haubo Bojesen Christensen. 2017. lmerTest package: tests in linear mixed effects models. *Journal of statistical software* 82, 13 (2017).

[19] Shinichi Nakagawa and Holger Schielzeth. 2013. A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in ecology and evolution* 4, 2 (2013), 133–142.

[20] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys. In *annual meeting of the Florida Association of Institutional Research*, Vol. 177. 34.

[21] Dhia Elhaq Rzig, Foyzul Hassan, Chetan Bansal, and Nachiappan Nagappan. 2022. Characterizing the Usage of CI Tools in ML Projects. In *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 69–79.

[22] Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. 2022. Investigating the impact of continuous integration practices on the productivity and quality of open-source projects. In *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 137–147.

[23] Diego Saraiva, Daniel Alencar Da Costa, Uirá Kulesza, Gustavo Sizílio, José Gameleira Neto, Roberta Coelho, and Meiyappan Nagappan. 2023. Unveiling the Relationship Between Continuous Integration and Code Coverage. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 247–259.

[24] Simon Sheather. 2009. *A modern approach to regression with R*. Springer Science & Business Media.

[25] Donald L Thistlethwaite and Donald T Campbell. 1960. Regression-discontinuity analysis: An alternative to the ex post facto experiment. *Journal of Educational psychology* 51, 6 (1960), 309.

[26] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. 805–816.

[27] Zhiyuan Wan, Xin Xia, David Lo, and Gail C Murphy. 2019. How does machine learning change software development practices? *IEEE Transactions on Software Engineering* 47, 9 (2019), 1857–1871.

[28] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2019. Studying software engineering patterns for designing machine learning systems. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 49–495.

[29] Yasuhiro Watanabe, Hironori Washizaki, Kazunori Sakamoto, Daisuke Saito, Kiyoshi Honda, Naohiko Tsuda, Yoshiaki Fukazawa, and Nobukazu Yoshioka. 2021. Preliminary literature review of machine learning system development practices. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 1407–1408.

[30] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A Gerosa. 2020. Effects of adopting code review bots on pull requests to oss projects. In *2020 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 1–11.

[31] Mairieli Wessel, Joseph Vargovich, Marco A Gerosa, and Christoph Treude. 2023. Github actions: the impact on the pull request process. *Empirical Software Engineering* 28, 6 (2023), 131.

[32] Daniel S Wilks. 2011. *Statistical methods in the atmospheric sciences*. Vol. 100. Academic press.

[33] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 60–71.