Are Machine Learning Projects Slower to Deliver? An Empirical Study of Pull Request Delivery Time

João Helis Bernardo
Federal Institute of Rio Grande do Norte (IFRN)
Santa Cruz - RN, Brazil
joao.helis@ifrn.edu.br

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

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

Abstract

How quickly merged pull requests (PRs) reach end users is a fundamental measure of software delivery efficiency. This paper investigates whether PR delivery time, defined as the interval between a PR merge and its inclusion in an official release, differs between Machine Learning (ML) and non-ML open-source projects. Analyzing 58 GitHub projects (27 ML, 31 non-ML), we find that ML projects experience significantly longer delivery times (median 68 vs. 40 days) and longer end-to-end PR lifetimes, from submission to release (81 vs. 49 days), despite statistically similar time to merge. Contrary to what one might expect, the bottleneck is not the code review phase but the post-merge release process. ML projects also release roughly half as frequently as non-ML projects (1.47 vs. 2.98 releases/year). Regression models reveal that the main predictors of delivery time are the same in both project types, suggesting that ML-specific delays reflect general deficiencies in release cadence and coordination rather than domain-specific technical constraints.

Keywords

Delivery Time, Machine Learning, Release Engineering

1 Introduction

How quickly merged code reaches end users is a fundamental measure of a software project's delivery efficiency. *PR delivery time*, defined as the interval between when a pull request (PR) is merged and when its changes appear in an official release, captures this dimension directly [5, 6, 9]. Prior work has shown that this metric varies widely across open-source projects and is shaped by factors such as queue position, contributor experience, and release cadence. Even in projects that adopt Continuous Integration (CI) to automate building and testing, significant post-merge delays can persist before changes are delivered to users.

Machine Learning (ML) projects introduce additional complexity into this picture. Training pipelines may take hours or days, datasets must be versioned and validated, and testing often involves non-deterministic model outputs. These demands extend well beyond what standard CI pipelines are designed to handle and can directly delay the post-merge steps required to prepare a stable release. Recent work has shown that ML projects already differ from traditional open-source software in measurable ways: they exhibit lower test coverage and longer build durations [7], and distinct reviewer engagement patterns [4]. A practitioner survey further reveals that automated deployment and release management are among the main challenges for ML teams [3]. Together, these findings raise a concrete question: *do ML projects take longer to deliver merged pull requests to end users than non-ML projects?*

No prior study has directly compared PR delivery time between ML and non-ML open-source projects. Understanding whether such a gap exists, whether it stems from code review or from the post-merge release phase, and what factors drive it is essential for guiding ML teams toward more effective release engineering practices.

This paper addresses this gap by presenting an empirical study of PR delivery dynamics in 58 open-source GitHub projects (27 ML and 31 non-ML). Our investigation focuses on three research questions:

- **RQ1. How does PR delivery time in ML projects compare to non-ML projects?** We quantify whether ML projects experience longer PR delivery times and locate where in the PR lifecycle the main differences occur.
- **RQ2. How do PR submission and evaluation patterns differ in the release dynamics between ML and non-ML projects?** We analyze whether ML and non-ML projects differ in PR submission behavior, evaluation outcomes, and release frequency.
- **RQ3. What are the key factors affecting PR delivery time in ML and non-ML projects?** We identify and compare the factors most strongly influencing PR delivery time to assess whether the determinants are domain-specific or reflect general software development dynamics.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 describes our research methodology. Sections 4 and 5 present results and discuss implications. Section 6 addresses threats to validity, and Section 7 concludes the paper.

2 Related Work

PR delivery time and release engineering. Prior work on software delivery latency has established that queue position, contributor experience, and release cadence are key predictors of how quickly fixed issues and merged PRs appear in a release [9, 10]. Gousios et al. [12] characterized the pull-based development model, while Bernardo et al. [5, 6] found that CI adoption does not consistently accelerate PR delivery, with only half of the studied projects benefiting. Our work extends this line by comparing PR delivery time between ML and non-ML projects for the first time, locating the delivery gap in the post-merge release phase rather than in code review.

CI in open-source software. Hilton et al. [15] found that CI-enabled projects release more frequently and integrate contributions faster. Bernardo et al. [4, 7] showed that ML projects exhibit lower test coverage, longer build durations, and distinct reviewer

engagement patterns compared to non-ML projects, while a practitioner survey [3] identified release management and automated deployment as major barriers. None of these studies directly measured whether such differences manifest as longer PR delivery times.

ML software engineering. Sculley et al. [23] identified hidden technical debt specific to ML systems, including data entanglement and pipeline fragility. Amershi et al. [1] found model deployment and monitoring to be particularly challenging stages, and Zimelewicz et al. [27] report limited MLOps adoption with teams struggling to integrate model serving into CI/CD pipelines. These findings motivate our investigation into whether ML-specific constraints translate into measurably longer PR delivery times.

3 Research Methodology

In this section, we describe the selection of studied projects, the database construction, and the design of each research question.

3.1 Studied Projects

We started from a curated dataset of 185 open-source projects (93 ML and 92 non-ML) hosted on GitHub, selected based on active GitHub Actions CI usage and balanced ML/non-ML representation [4]. From this dataset, we identified 180 projects (90 ML and 90 non-ML) with release tags available through the GitHub API. Figure 1 illustrates the filtering steps that led to our final sample.

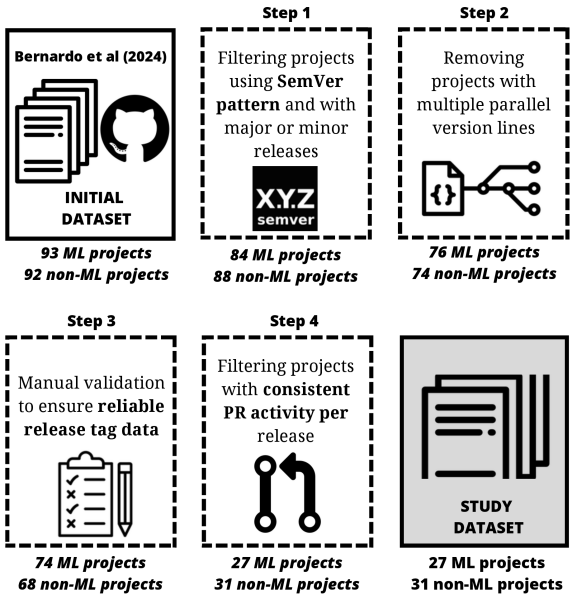


Figure 1: Project selection process.

Step 1 – Filtering by Semantic Versioning and Release Type. We verified adherence to Semantic Versioning (SemVer) [20] and retained only projects using valid SemVer tags. Patch releases and non-user-intended tags (pre-releases, alpha, beta) were excluded. After this step, 84 ML and 88 non-ML projects remained.

Step 2 – Removing Projects with Multiple Parallel Version Lines. Projects maintaining multiple parallel version lines (e.g., `v1.x` and `v2.x` concurrently) were excluded, leaving 76 ML and 74 non-ML projects.

Step 3 – Manual Validation. The remaining projects were manually reviewed to ensure data quality. Eight projects were excluded for having too few valid releases, excessive non-SemVer tags, or extremely sparse release histories. This left 74 ML and 68 non-ML projects.

Step 4 – Focusing on Projects with Active PR Activity. We required each project to exhibit sustained PR activity, defined as a median of at least one merged PR per release. The final sample comprises **58 projects: 27 ML and 31 non-ML**.

3.2 Data Collection

We collected our dataset using an updated version of the `ml-ci-project-miner` tool, provided in the replication package of Bernardo et al. [2]. This Java-based mining tool automates the retrieval and processing of GitHub repository metadata through the GitHub API, supporting the extraction of tags, releases, commits, PRs, and general repository metadata. Data collection was performed in June 2024.

For each repository, we collected: general repository metadata (forks, stars, contributors, language), commit data, release tag data, and PR metadata (state, author association, changed files, additions, deletions, commits, comments, participants, and timestamps). From PR timestamps, we derived *merge time* (creation to merge) and *rejection time* (creation to close for non-merged PRs).

Pull Request–to–Release Association. To measure delivery time, we linked merged PRs to their first subsequent release tag via a release interval analysis based on chronological SemVer ordering. For each project, the time span between two consecutive tags defined a *release interval*. A merged PR was linked to a release interval if its merge date fell within that interval. We then computed: (i) **Delivery time**: days between PR merge date and release date; and (ii) **PR lifetime**: days between PR creation and release date.

3.3 Research Questions Design

RQ1. How does PR delivery time in ML projects compare to non-ML projects? The basic PR lifecycle comprises two phases: t_1 (merge phase: submission to merge) and t_2 (delivery phase: merge to release). The sum $t_1 + t_2$ defines the total PR lifetime. For each project, we calculated the median of these three metrics across all releases. To compare ML and non-ML projects, we applied the Mann-Whitney-Wilcoxon (MWW) test [25] and Cliff's Delta [8] to assess both statistical significance and practical magnitude. Following Romano et al. [21], we interpret Cliff's Delta as: negligible (<0.147), small (<0.33), medium (<0.474), and large (≥ 0.474). We present results using annotated boxplots [26].

RQ2. How do PR submission and evaluation patterns differ in the release dynamics between ML and non-ML projects? We computed and compared: PR submission rate (median submitted PRs per release), PR acceptance/rejection rates, merge-to-reject ratio, and release cadence (median releases per year). Metrics were aggregated at the project level. We use MWW tests [25] and Cliff's Delta [8] for comparisons.

RQ3. What are the key factors affecting PR delivery time in ML and non-ML projects? We use multiple regression modeling with Ordinary Least Squares (OLS) to examine relationships between a set of explanatory variables and PR delivery time [14]. We fit one model per project to capture project-specific relationships. Tables 1 and 2 define the explanatory variables. The modeling process follows Harrell Jr. [13] and is summarized in Figure 2.

Table 1: Explanatory variables: Contributor; Process and Workflow.

Dim.	Variable	Type	Definition / Rationale
Contributor	Contributor Experience	Num.	No. of previously released PRs by the same author [24].
	Contributor Integration	Num.	Avg. delivery time of the author's previous PRs [9].
	Author Assoc. (MEMBER)	Bool.	Author is a repository member.
	Author Assoc. (CONTRIBUTOR)	Bool.	Author is a previous contributor.
	Author Assoc. (NONE)	Bool.	Author has no prior association.
Process & Workflow	Release Type	Cat.	Type of release (major, minor) in which the PR was included.
	Queue Rank	Num.	Position of the PR in the release merge queue [9].
	Merge Workload	Num.	No. of PRs waiting to be merged at submission time.
	Merge Time	Num.	Days between PR submission and merge.

Table 2: Explanatory variables: PR Size and Complexity; Collaboration.

Dim.	Variable	Type	Definition / Rationale
PR Size & Complexity	Commits Count	Num.	No. of commits in the PR [19].
	Changed Files	Num.	No. of changed files in the PR.
	Churn	Num.	Added + deleted lines in the PR [17].
Collaboration	No. of Comments	Num.	No. of comments on the PR [11].
	Participants Count	Num.	No. of unique participants in the PR discussion.

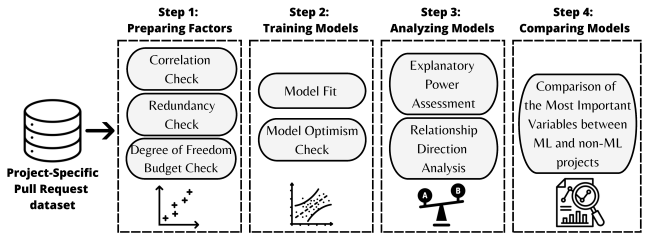


Figure 2: Overview of the process used to build our explanatory models.

Step 1: Preparing Factors. We address potential collinearity using the *redun* function from the *rms* R package [10]. Variables with $R^2 \geq 0.9$ are considered redundant and discarded. We then apply variable clustering [22] and retain only one representative variable from clusters with $|\rho| > 0.7$ correlation. Finally, we estimate the degrees of freedom (D.F.) budget as $\frac{n}{10}$ [13], preventing overfitting.

Step 2: Training Models. We assess model fit using R^2 [16] and compute the *optimism-reduced* R^2 via 1,000 bootstrap samples [18]. We retain only models with optimism-reduced $R^2 > 0.5$. Of 58 evaluated models, 21 (11 ML, 10 non-ML) met this threshold.

Step 3: Analyzing Models. We apply the Wald χ^2 test to evaluate each variable's contribution to explained variance. We use the `Predict` function from the `rms` package to plot the effect of the most influential variables on PR delivery time.

Step 4: Comparing Models. We rank variables by Wald χ^2 values per project and compare their distributions between ML and non-ML projects.

4 Results

RQ1. How does PR delivery time in ML projects compare to non-ML projects?

Figure 3 presents a comparative overview of PR lifecycle metrics for the 27 ML and 31 non-ML projects, reporting p-values and Cliff's Delta (d) values.

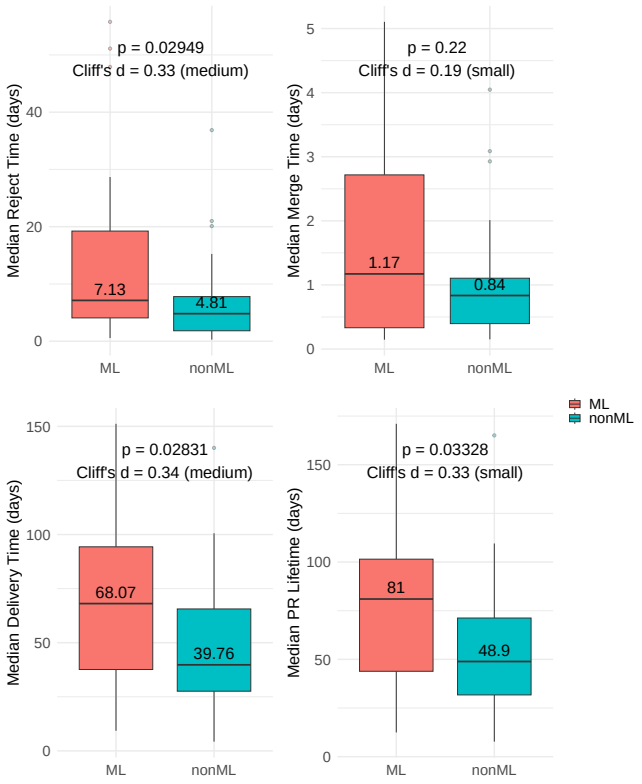


Figure 3: Comparative overview of PR lifecycle metrics in ML and non-ML projects.

The median rejection time in ML projects was 7.13 days vs. 4.81 days in non-ML projects ($p = 0.0295$, Cliff's $d = 0.33$, medium).

This suggests that rejected PRs remain open longer in ML projects, possibly reflecting longer evaluation periods.

No statistically significant difference was observed in merge time (1.17 vs. 0.84 days, $p = 0.22$, $d = 0.19$, small), indicating that the integration of accepted changes proceeds at a similar pace in both project types.

ML projects exhibited a significantly longer median delivery time than non-ML projects (68.07 vs. 39.76 days, $p = 0.0283$, Cliff's $d = 0.34$, medium). This delay may reflect additional steps in ML release workflows (model retraining, dataset updates, validation), though further investigation is needed. The cumulative effect is visible in overall PR lifetime: ML projects had a median of 81.00 days vs. 48.90 days in non-ML projects ($p = 0.0333$, $d = 0.33$, small).

Summary (RQ1): ML projects experience significantly longer delivery times and overall PR lifetimes despite comparable merge durations. The primary bottleneck lies in the post-merge release phase, not in code review or integration.

RQ2. How do PR submission and evaluation patterns differ in the release dynamics between ML and non-ML projects?

Non-ML projects tend to receive more PRs per release (median 58) than ML projects (median 33), though the difference is not statistically significant ($p = 0.1677$, $d = -0.21$, small). Figure 4 illustrates this.

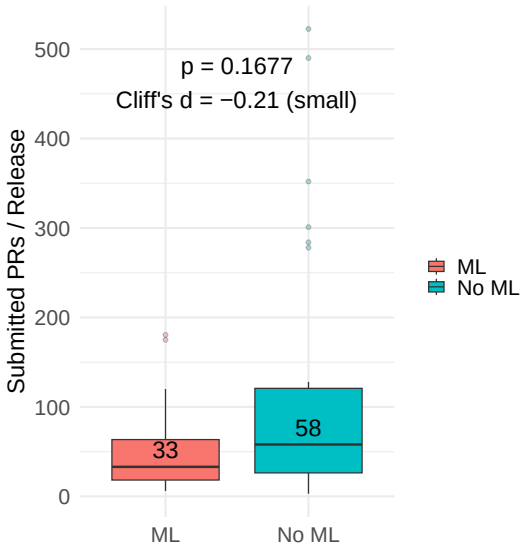


Figure 4: Number of submitted PRs per release in ML and non-ML projects.

Regarding PR outcomes (Figure 5), ML projects exhibit a significantly lower rejection rate (12.27% vs. 19.36%, $p = 0.007163$, $d = -0.41$, medium) and a slightly higher merge rate (84.12% vs. 79.56%, $p = 0.007853$, $d = 0.41$, medium).

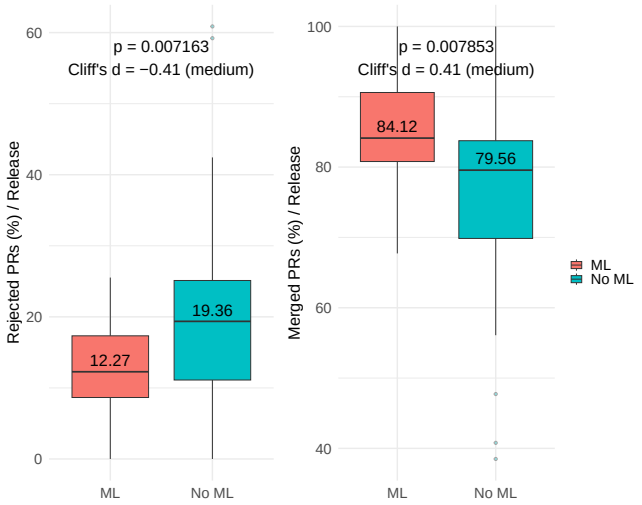


Figure 5: Percentage of rejected and merged PRs per release.

The merge-to-reject ratio (Figure 6) is significantly higher in ML projects (median 6.75 vs. 4, $p = 0.007855$, $d = 0.41$, medium), meaning roughly one rejection per seven merges in ML vs. one per four in non-ML. This may reflect stricter PR submission culture in ML contexts.

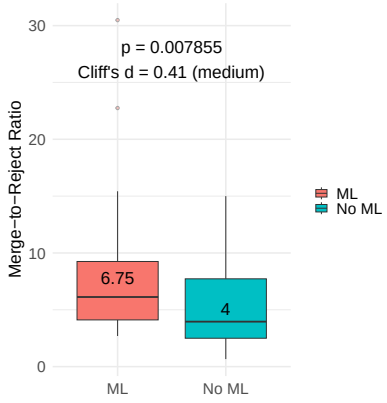


Figure 6: Merge-to-reject ratio per release.

Finally, ML projects have a significantly lower release cadence: 1.47 releases/year (one every ~8 months) vs. 2.98 releases/year (one every ~4 months) in non-ML ($p = 0.009371$, $d = -0.40$, medium).

Summary (RQ2): ML projects merge a higher proportion of PRs and reject fewer per release (merge-to-reject ratio: 6.75 vs. 4), but release significantly less frequently (1.47 vs. 2.98 per year), which directly contributes to longer delivery times.

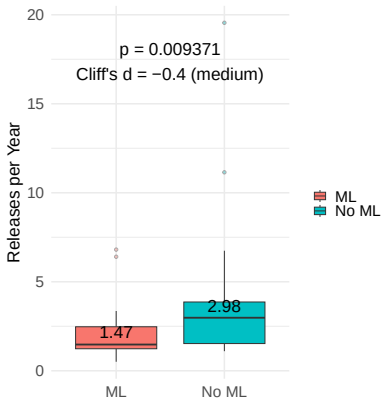


Figure 7: Release frequency (releases per year).

RQ3. What are the key factors affecting PR delivery time in ML and non-ML projects?

Our models achieved a median R^2 of 0.60 for both ML and non-ML projects. After correcting for optimism, both groups converged to a median corrected R^2 of 0.58, with median optimism estimates of 0.01, indicating low overfitting. Of 58 fitted models, 21 (11 ML, 10 non-ML) met the retention threshold of optimism-reduced $R^2 > 0.5$; the comparisons that follow are based on this subset.

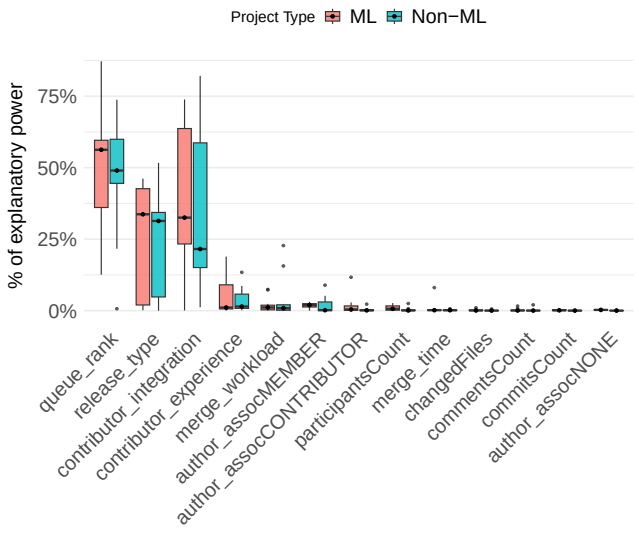


Figure 8: PR delivery time explanatory power by variable and project type.

Figure 8 shows that three variables account for most explained variance: `queue_rank`, `release_type`, and `contributor_integration`. Among these, `queue_rank` clearly stands out, with median explanatory power around 50% for both project types. The remaining predictors (changed files, comments, commits, participants, merge workload) each explain less than 5% on average.

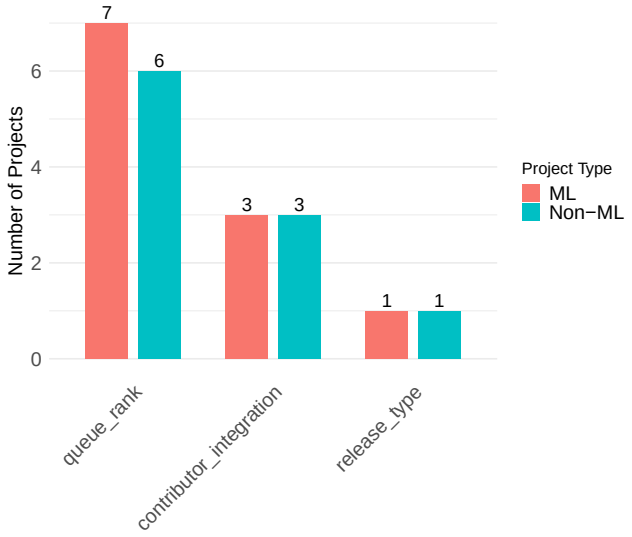
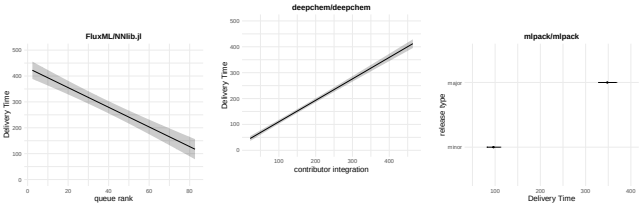


Figure 9: Top predictor per project based on percentage of variance explained.

Figure 9 shows the top predictor per project. `queue_rank` dominates (7 ML, 6 non-ML projects). `contributor_integration` is second (3 ML, 3 non-ML), and `release_type` appears as the top predictor in one ML and one non-ML project.

Figure 10 reveals the direction of effects: `queue_rank` has a negative association with delivery time (PRs merged later in the cycle reach the next release faster); `contributor_integration` has a positive relationship (contributors whose past PRs took longer tend to experience longer delivery times again); and major `release_type` is associated with longer delivery times than minor releases.



(a) queue_rank **(b) contributor_integration** **(c) release_type**

Figure 10: Relationship between top predictors and delivery time.

Summary (RQ3): PR delivery time is primarily driven by workflow-related factors: `queue_rank` (position in the release cycle), `contributor_integration` (historical delivery performance), and `release_type`. These drivers are largely shared across ML and non-ML projects within the retained subset, suggesting that longer delivery times in ML may not be uniquely tied to ML-specific workflows.

5 Discussion

Interpreting Longer PR Lifetimes in ML Projects

Our RQ1 findings show that ML projects have longer delivery times than non-ML projects. The lower release cadence observed in RQ2 directly contributes to extended PR lifetimes. This slower cycle likely reflects deeper technical challenges: prior work has shown that ML projects tend to exhibit lower test coverage and longer build durations compared to non-ML projects [7], driven by complex testing and high infrastructure demands such as data validation and model training.

A practitioner survey on CI in ML projects [3] corroborates this picture: practitioners report that limited automated deployment is a recurring barrier to delivery, with ML teams frequently struggling to integrate model serving into their CI/CD pipelines. Zimelewicz et al. [27] further report limited MLOps adoption, frequent deployment of models as separate services, and substantial difficulties with production infrastructure integration. Together, these findings provide converging evidence that the release process constitutes a major bottleneck for ML projects. Teams should integrate data validation, dependency management, and automated model deployment as core steps in the CI/CD pipeline.

Interpreting the Drivers of Delivery Time

The regression analysis in RQ3 identifies a small set of variables that consistently explain most of the variance in PR delivery time, primarily related to workflow management and contributor behavior rather than project type or PR content.

Queue position and workflow timing. `Queue_rank` alone explains around half of the explained variance in most models. While this effect is partly mechanical (PRs merged later have less time until release), it points to the need for coordinated release management and predictable integration cadence.

Contributor history. `Contributor_integration` shows a positive relationship with delivery time: contributors whose previous PRs took longer to be released tend to experience longer delivery times again. Teams can mitigate this inertia through improved onboarding, clear contribution guidelines, and pairing contributors with experienced integrators.

Release policy. `Release_type` influences delivery time, with major releases generally taking longer, reflecting broader integration scope. Common PR-level metrics (changed files, comments, commits) remain minor contributors once workflow-level factors are controlled.

Implications for ML projects. Since ML projects typically exhibit lower release cadence, adopting more frequent and smaller release batches could substantially reduce delivery times. The regression models suggest ML-specific delays arise primarily from general workflow coordination rather than ML-specific technical constraints.

6 Threats to Validity

Construct validity. Our sample covers diverse domains within both ML and non-ML categories, but we did not stratify results by domain (e.g., deep learning frameworks, scientific computing), which may influence PR delivery dynamics. Additionally, the high explanatory power of `queue_rank` is partly mechanical: PRs merged closer to a release cut naturally have less time until release, inflating its correlation with delivery time. Furthermore, the PR-to-release association relies on chronological ordering rather than commit reachability verification, which may introduce false positives if a PR's changes were not effectively included in the linked release tag.

Internal validity. As an observational study, the relationships found between variables and PR lifetime cannot be interpreted as causal. Confounding factors (governance style, release policy changes, organizational practices) may influence both predictors and outcomes. Although we reduced multicollinearity, omitted relevant variables (e.g., PR priority) could bias coefficients.

External validity. Our dataset consists of GitHub-hosted projects following a PR-based workflow, which may limit generalizability to other platforms (GitLab, Bitbucket). We filtered out inactive projects or those with sparse release histories, biasing the sample toward more active and mature projects. The ML subset contains open-source frameworks, applied tools, and libraries; results may not generalize to proprietary ML systems. Furthermore, our delivery time measure assumes that release tags represent the moment changes reach users. In projects that practice Continuous Deployment, merged PRs are delivered to users immediately after merge, making the tag an inadequate proxy for delivery; such projects were excluded from our sample by construction.

7 Conclusion

We investigated PR delivery dynamics in ML and non-ML open-source projects, analyzing data from 58 GitHub projects (27 ML and 31 non-ML). Our key findings are: (i) ML projects exhibit significantly longer delivery times (68 vs. 40 days) and overall PR lifetimes (81 vs. 49 days) than non-ML projects, despite statistically similar time to merge; (ii) ML projects receive fewer PRs per release and reject a smaller proportion, resulting in a higher merge-to-reject ratio (6.75 vs. 4) and a slower release cadence (1.47 vs. 2.98 releases/year); and (iii) three variables explain most of the variance in PR delivery time in both project types: `queue_rank`, `contributor_integration`, and `release_type`.

These findings suggest that ML-specific delivery delays stem less from domain-specific technical constraints than from general deficiencies in release coordination and cadence. For ML projects, shortening post-merge delays through more frequent releases and improved MLOps practices could substantially improve delivery efficiency. Future work should investigate domain-specific effects across ML sub-domains and examine how release queue management and contributor experience interact in more complex team settings.

Artifact Availability

The replication package, including data collection scripts and analysis scripts, is available at: https://github.com/joaohelis/ml-ci-project-miner [2].

Acknowledgements

Many thanks to INES.IA (www.ines.org.br), CNPq (408817/2024-0, 311749/2025-94-0) and CAPES (88887.186328/2025-00).

References

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] João Helis Bernardo, Daniel Alencar da Costa, Sérgio Queiroz de Medeiros, and Uira Kulesza. 2024. *How do Machine Learning Projects use Continuous Integration Practices? An Empirical Study on GitHub Actions.* doi:10.5281/zenodo.10637336

[3] João Helis Bernardo, Daniel Alencar da Costa, Filipe Roseiro Cogo, Sérgio Queiróz de Medeiros, and Uirá Kulesza. 2025. Continuous Integration Practices in Machine Learning Projects: The Practitioners' Perspective. *arXiv preprint arXiv:2502.17378* (2025).

[4] João Helis Bernardo, Daniel Alencar da Costa, Sérgio Queiroz de Medeiros, and Uirá Kulesza. 2025. Exploring the Impact of GitHub Actions on Pull Request Reviews in Machine Learning Projects. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 650–656.

[5] 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.

[6] 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.

[7] João Helis Bernardo, Daniel Alencar Da Costa, Sérgio 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 21st International Conference on Mining Software Repositories*. 665–676.

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

[9] Daniel Alencar da Costa, Shane McIntosh, Uirá Kulesza, and Ahmed E. Hassan. 2016. The Impact of Switching to a Rapid Release Cycle on the Integration Delay of Addressed Issues: An Empirical Study of the Mozilla Firefox Project. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, USA, 374–385.

[10] Daniel Alencar Da Costa, Shane McIntosh, Uira Kulesza, Ahmed E Hassan, and Surafel Lemma Abebe. 2018. An empirical study of the integration time of fixed issues. *Empirical Software Engineering* 23, 1 (2018), 334–383.

[11] Emanuel Giger, Martin Pinzger, and Harald Gall. 2010. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, 52–56.

[12] 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.

[13] Frank Harrell. 2015. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning: data mining, inference and prediction* (2 ed.). Springer.

[15] 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.

[16] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

[17] Yujuan Jiang, Bram Adams, and Daniel M German. 2013. Will my patch make it? and how fast? case study on the linux kernel. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 101–110.

[18] Shane Mcintosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2016. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Softw. Engg.* 21, 5 (2016), 2146–2189.

[19] Nachiappan Nagappan and Thomas Ball. 2005. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 284–292.

[20] Tom Preston-Werner. 2013. Semantic Versioning 2.0.0. https://semver.org/. Accessed: 2025-08-09.

[21] 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.

[22] WS Sarle. 1990. The VARCLUS Procedure. SAS/STAT User's Guide. SAS Institute. *Inc., Cary, NC, USA* (1990).

[23] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, Vol. 28.

[24] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M Ibrahim, Masao Ohira, Bram Adams, Ahmed E Hassan, and Ken-ichi Matsumoto. 2010. Predicting reopened bugs: A case study on the eclipse project. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*. IEEE, 249–258.

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

[26] David F. Williamson, Robert A. Parker, and Juliette S. Kendrick. 1989. The box plot: A simple visual method to interpret data. *Annals of Internal Medicine* 110 (1989), 916–921.

[27] Eduardo Zimelewicz, Marcos Kalinowski, Daniel Mendez, Görkem Giray, Antonio Pedro Santos Alves, Niklas Lavesson, Kelly Azevedo, Hugo Villamizar, Tatiana Escovedo, Helio Lopes, et al. 2024. Ml-enabled systems model deployment and monitoring: Status quo and problems. In *International Conference on Software Quality*. Springer, 112–131.