Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Quantifying SZZ-Induced Evaluation Bias in Just-in-Time Defect Prediction

Peter Bludau

peter.bludau@tum.de

Technical University of Munich

Alexander Pretschner

Technical University of Munich

Research Article

Keywords: Just-in-Time Defect Prediction, SZZ, Defect Dataset Quality, Empirical Evaluation

Posted Date: June 1st, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9846265/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Software Quality Journal on October 22nd, 2026. See the published version at https://doi.org/10.1007/s11219-026-09796-x.

Quantifying SZZ-Induced Evaluation Bias in Just-in-Time Defect Prediction

Peter Bludau[1*] and Alexander Pretschner[1]

[1]Technical University of Munich, Munich, Bavaria, Germany.

*Corresponding author(s). E-mail(s): peter.bludau@tum.de;
Contributing authors: alexander.pretschner@tum.de;

Abstract

Just-in-time defect prediction identifies defect-inducing commits at commit time to focus quality assurance efforts, but its validity depends on the quality of underlying defect datasets, typically generated using SZZ-based algorithms. Because the correctness of these datasets is discussed frequently, this study sets out to quantify how residual SZZ errors propagate into just-in-time defect prediction evaluation. We construct a manually validated ground-truth sample covering 420 commits across 6 open-source projects and provide a stratified, weighted evaluation protocol for estimating population-level performance. Using this sample, we evaluate PR-SZZ-generated datasets and reassess state-of-the-art model performance. We find that PR-SZZ generates wrong defect labels for approximately 12% of commits, mainly through false positives. When just-in-time defect prediction models learned on these defect datasets are evaluated against manual ground truth rather than PR-SZZ labels, average model recall drops from 0.72 to 0.64 and precision from 0.27 to 0.19, indicating that SZZ-based evaluations can substantially misrepresents real model performance. We also find that previously reported performance improvements gained by feature engineering only partly persist under manual evaluation, suggesting that added model capacity may capture systematic labeling errors rather than genuine defect-inducing behavior, particularly as feature characteristics can correlate with SZZ mechanisms. Overall, our findings highlight the need for reliable ground-truth inference and rigorous evaluation practices.

Keywords: Just-in-Time Defect Prediction, SZZ, Defect Dataset Quality, Empirical Evaluation

1 Introduction

Just-in-Time defect prediction (JIT-DP) aims to identify defect-inducing commits at commit time, enabling practitioners to prioritize review and testing efforts. To train and evaluate such models, historical defect datasets are typically constructed for the project under investigation, often using the SZZ algorithm [26]. SZZ traces backward from bug-fixing commits through the version history to identify the commits that likely introduced the corresponding defects. It then generates a binary defect label for every commit. These labels support standard machine learning workflows: commits are split into training and test sets, classification models learn to distinguish defective from clean commits based on the training set, and their predictive performance is evaluated on the unseen test data.

Because the SZZ algorithm automatically infers defect-inducing commits from project history, its labels are not always accurate. In some cases, SZZ may misclassify clean commits as defect-inducing or fail to identify commits that actually introduced defects. To address these limitations, various variants of the algorithm have been proposed in the literature to produce higher-quality ground truth for defect prediction studies [14, 29, 5, 4, 19, 2, 28, 27, 7, 22]. Despite these efforts to improve label accuracy and dataset completeness, the fundamental limitation of SZZ-based datasets, and consequently all research that is based on these datasets, remains: automatically inferred labels are imperfect [9]. Prior studies have repeatedly shown that standard SZZ generates a number of false defect-inducing links, that multiple SZZ variants reduce but do not eliminate noise, and that the accuracy of downstream defect prediction models may depend on the quality of these labels [26, 10, 4, 8, 28]. Label noise can distort both the estimated performance of models and the inferred relationships between features and defect proneness. Nonetheless, virtually all prior work on JIT-DP relies on SZZ-generated labels, largely because manually validated ground truth at commit level is scarce and expensive to obtain. Moreover, to the best of our knowledge only little research exists that quantifies the influence of bias in defect dataset labels on the model results and no prior study has systematically evaluated the impact of SZZ misclassifications on JIT-DP performance using manually validated ground truth.

In this work, we quantify the impact of SZZ-induced evaluation bias on reported JIT-DP performance and examine the implications for the field. Understanding the extent and consequences of residual labeling errors is essential for assessing the true effectiveness and practical applicability of state-of-the-art JIT-DP models. This motivates the following research questions:

RQ1: What is the direction, magnitude, and project-dependence of PR-SZZ label noise?
RQ2: How much evaluation bias do PR-SZZ labels introduce into standard JIT-DP metrics?
RQ3: To what extent are previously reported improvements and contributions affected by label noise?

To answer these questions, we manually validate a stratified sample of commits from PR-SZZ-generated datasets [2] and evaluate both PR-SZZ labels and JIT-DP

model predictions against the resulting manual reference labels. Our contributions are as follows:

- We construct a manually validated reference commit set covering the period from 1 January 2021 to 30 June 2021, comprising 420 commits across six projects. We introduce a stratified, weighted evaluation protocol that supports population-level estimates under a constrained labeling budget and apply it to both label-error estimation and model-performance evaluation.
- We quantify the residual error profile of PR-SZZ and show that its misclassifications are directional, project-dependent, and concentrated among commits labeled as defect-inducing.
- We demonstrate that these errors materially bias standard JIT-DP evaluation metrics, inflating precision and F1 when PR-SZZ labels are treated as evaluation ground truth.
- We revisit the feature-level improvements reported by Bludau et al. [1] by comparing a model trained on their extended feature set with a baseline model. We show that gains observed under PR-SZZ-based evaluation do not persist under manually validated labels, suggesting that richer feature sets may partly capture systematic labeling artifacts.
- We derive a taxonomy of qualitative failure modes and present practical guidelines for more reliable ground-truth construction and deployment of JIT-DP systems.

Our results show that PR-SZZ mislabels approximately 12% of commits, with a pronounced bias towards false positives (commits incorrectly marked as defect-inducing). These misclassifications can substantially distort apparent performance of JIT-DP models. On average, the model's recall decreases from 0.72 under PR-SZZ labels to 0.64 under manual ground truth, while precision drops from 0.27 to 0.19. Thus, SZZ-based evaluation overstates both the ability of models to identify truly defect-inducing commits and their practical precision. These findings confirm and quantify a long-standing concern: models trained and evaluated on SZZ-derived datasets can appear more effective than they are under manual validation. Consequently, the feature-engineering gains reported by Bludau et al. [1] disappear when evaluated against manually validated labels. This suggests that feature improvements that appear beneficial under SZZ-based evaluation may not necessarily translate into genuine improvements in real-world defect detection, especially when model features are correlated with the inner workings of the SZZ algorithm.

2 Outline

In this study, we investigate the correctness of SZZ-labeled datasets and their influence on reported performance. Section 3 provides an overview of relevant research on SZZ and JIT-DP and situates this work within the existing literature. Section 4 describes the data and evaluation foundation this study builds on, precisely defect datasets created by the PR-SZZ algorithm [2] and subsequent evaluation of feature sets and modeling techniques using these datasets [1]. Consequently, it introduces the

methodology and approach to answer the research questions posed in the introduction. The study combines (i) an automatic evaluation on the full 2021-H1 population using PR-SZZ labels and (ii) a manual evaluation on a stratified sample of 420 commits. In Section 5, we outline the study results following the described methodology. All presented findings are based on the validated dataset and weighted performance metrics and are aligned with the posed research questions. Lastly, Section 6 discusses qualitative findings, introduces a taxonomy of common error classes, and comments on the implications for research and industry. We conclude this study in Section 8 by summarizing the main findings and their implications for the field of JIT-DP.

3 Background

JIT-DP is commonly formulated as a binary classification problem, where a model predicts whether a given commit is defect-inducing. Existing approaches typically rely on standard machine learning or deep learning techniques, trained and evaluated on defect datasets and features derived from software repositories. This section reviews the main strands of related work in this area. First, we discuss algorithms for constructing defect datasets used in JIT-DP. We then summarize the features commonly extracted for model training. Finally, we examine studies that investigate how defect dataset quality affects the reported performance of defect prediction models.

3.1 Defect Datasets

Since defect-inducing commits are rarely documented explicitly, most JIT-DP studies construct their ground truth retrospectively using the SZZ algorithm and its variants. In its original form, SZZ first links bug reports to defect-fixing commits and then traces the lines modified by those fixes back through the version control history to identify candidate defect-inducing commits in a two-step process [26]:

- **First, defect-fixing commits in the version control system are identified**, typically by searching for defect-related keywords in commit messages and linking these commits to defect reports from the issue tracking system, such as Jira or Github Issues. To this end, bug tickets are collected, and candidate fixing commits are identified. Some algorithms construct the list of candidates solely from commit message, for example by relying on mentions of bug ticket identifiers and keywords [23, 11]. However, such datasets often contain many commits that are falsely labeled as defect-fixing commits. The latest commit in this candidate set is then selected as the fixing commit. This process implies that a defect can only appear in the dataset if it has been reported in the issue tracking system.
- **Second, defect-inducing commits are identified**. For each fixing commit, the algorithm traces the modified lines back through the version control history to determine when they were last changed. The algorithm computes the difference between each fixing commit and its direct parent to identify the lines changed by the fix. It then uses the version control's *blame* feature to retrieve the most recent commits that previously modified those lines. These commits form the list of defect-inducing commits.

The resulting defect datasets are subsequently used in JIT-DP studies as the foundation for model training and evaluation. As a result, the quality of JIT-DP models depends directly on the correctness of the labels produced by this tracing process.

A large body of work has shown that this labeling process is inherently error-prone. Misclassifications can arise from imperfect links between issues and commits, tangled changes, refactorings, cosmetic edits, merge commits, and the line-based nature of blame-based tracing [9]. To address these shortcomings, several variants of SZZ have been proposed, most of them targeting the second phase of the algorithm. For instance, AG-SZZ reduces noise caused by cosmetic changes [14], MA-SZZ filters meta-changes [4], and RA-SZZ excludes refactorings from the candidate set of bug-inducing commits [20, 19]. Bludau and Pretschner later proposed PR-SZZ, which incorporates pull-request data to improve both the identification of bug-fixing commits and the filtering and selection of bug-inducing commits [2]. Subsequent studies introduced refinements based on work-item information [22], semantic or refactoring-aware change analysis [28, 21], and large language models for candidate assessment and traceability [27, 7]. More recently, Shi et al. move beyond the standard two-step SZZ procedure by constructing a knowledge graph over development artifacts and querying it to identify bug-inducing commits [24]. Collectively, these approaches represent important steps toward reducing label noise and improving the credibility of downstream JIT-DP evaluation.

3.2 Features Engineering

Besides ground-truth quality, the effectiveness of JIT-DP also depends on the features used to represent code changes. Prior work has established a strong baseline based on process, churn, diffusion, history, and developer-experience metrics, such as lines added and deleted, number of modified files, entropy of the change, prior modifications, and developer experience [18, 25, 13]. Later work extended these baselines with review-related metrics derived from modern code review workflows [15]. Together, these features form the core of most state-of-the-art metric-based JIT-DP models. Bludau and Pretschner extended these established features with two additional families of features [1]. The first captures properties of the surrounding software development workflow from pull requests. The second captures structural information from source-code changes using abstract syntax tree metrics, for example the number of added, removed, and updated nodes. When evaluated on their PR-SZZ-derived datasets, they report a significant improvement in predictive model performance using these additional features.

3.3 Evaluation of Label Noise Influence

The quality of SZZ-derived datasets remains subject to ongoing debate. This has direct implications for the study of feature effectiveness, since features cannot be evaluated independently of the quality of the labels on which models are trained and tested. If the ground truth contains systematic errors, models may learn patterns that reproduce the labeling behavior of SZZ rather than patterns that characterize truly defect-inducing changes. As a consequence, apparent improvements in predictive performance or shifts

in feature importance may reflect sensitivity to labeling artifacts rather than genuine improvements in defect prediction. Reliable ground truth is therefore essential not only for fair model evaluation, but also for drawing meaningful conclusions about which feature sets are useful in practice.

Herbold et al. [9] examined the state of practice in defect prediction data collection and found that inaccurate SZZ-based labels remain a major threat to validity. Their results suggest that problems in defect labeling can outweigh limitations caused by restricted feature availability, emphasizing that data quality is a central concern for defect prediction research. Although their study raises fundamental questions about the validity of conclusions drawn from JIT-DP models trained and evaluated on SZZ-derived datasets, it does not directly quantify how label quality affects predictive performance.

The study most closely related to ours is that of Fan et al. [8], who investigated how mislabeled changes produced by different SZZ variants affect JIT-DP models. Their results suggest that label noise does not always lead to large differences in conventional performance metrics, but can nevertheless increase wasted inspection effort and distort the interpretation of learned models. This is an important observation, as it indicates that noisy labels may bias practical conclusions even when aggregate predictive performance appears relatively stable. However, Fan et al. use defect labels produced by the best-performing SZZ variant at the time, RA-SZZ [19], as the ground truth for their evaluation. While this approach is reasonable, relying on an imperfect algorithm to construct the ground truth introduces bias, particularly because RA-SZZ itself remains prone to substantial noise. As a result, their evaluation may still underestimate the extent to which noise in SZZ-derived datasets affects the effectiveness, interpretability, and practical validity of state-of-the-art JIT-DP models.

4 Methodology

We set out to investigate the influence of SZZ label noise by manually creating a high-quality defect dataset and comparing it with SZZ-derived labels and model predictions. In this section, we introduce the data foundation and methodology used to conduct our study.

4.1 Study Context and Prior Artifacts

The data used in this study builds on the work of Bludau and Pretschner, who made the datasets, feature sets, and experimental artifacts from their studies available as replication packages. Using the PR-SZZ datasets introduced in [2] together with the feature-based evaluation framework established in [1] therefore provides a transparent and comparatively strong SZZ-based baseline for our analysis. Their published dataset contains commit-level defect information for six large open-source projects, covering the period January 2015 to December 2020 [1] extracted using PR-SZZ. The replication package for their subsequent work contains defect prediction features for every commit in these projects and descriptions of the model construction process.

[1] The projects are: `apache/airflow (Airflow)`, `angular/angular (Angular)`, `apache/calcite (Calcite)`, `jenkinsci/jenkins (Jenkins)`, `apache/kafka (Kafka)`, and `apache/pulsar (Pulsar)`.

These artifacts are well suited for our study because they allow prior JIT-DP results to be reassessed in a structured and reproducible setting. In particular, because the feature engineering study in [1] relies on the same PR-SZZ-derived datasets introduced in [2], it enables us to examine whether previously reported feature-level improvements reflect genuine predictive value or are influenced by residual labeling errors. We reuse the trained models from the 2015 to 2020 period and apply them to new, unseen commits between January and June 2021 (2021-H1), as 6-month periods are common evaluation time-frames in many other studies [17]. Lastly, we compare the resulting predictions both to automatically inferred PR-SZZ labels for this period and to our manually constructed ground-truth dataset.

The goal of our study is to assess the predictive performance and practical usability of JIT-DP models trained on SZZ-labeled data when applied to unseen commits and quantify the extent to which SZZ-derived labels may distort the measured performance of JIT-DP models when compared to a manually curated ground truth. Our work differs from existing validation studies in two important ways. First, rather than taking the output of one SZZ variant as the ground truth we construct a high-quality defect dataset for the six projects manually that is temporally disjoint from the training period (2021-H1). This enables a more unbiased analysis of error modes and concrete labeling errors. An important part of this analysis is to assess how residual PR-SZZ misclassifications affect commonly reported performance metrics (precision, recall, F1, and Area Under the ROC Curve (AUC)), and to what extent they inflate apparent model effectiveness. Second, we evaluate not only the influence of misclassifications on overall model performance, but also revisit previously reported feature-level improvements. This allows us to assess whether label noise affects conclusions about the usefulness of richer feature sets. In this sense, our study connects prior work on SZZ quality, JIT-DP feature engineering, and evaluation validity by examining how these factors interact in practice.

To answer RQ1 (What is the direction, magnitude, and project-dependence of PR-SZZ label noise?), we compare PR-SZZ labels to the manually validated ground truth for a stratified sample of commits from the 2021-H1 period. This allows us to quantify the overall misclassifications rate of PR-SZZ and to analyze the distribution of errors across different projects.

RQ2 (How much evaluation bias do PR-SZZ labels introduce into standard JIT-DP metrics?) is answered by evaluating model predictions (trained using the original datasets) against both PR-SZZ labels from the 2021-H1 period and the collected manual ground truth. This comparison quantifies the extent to which PR-SZZ misclassifications skews performance metrics.

Finally, RQ3 (To what extent are previously reported improvements and contributions affected by label noise?) is addressed by comparing the performance of models trained using the additional features proposed by Bludau et al. to a baseline model trained only on traditional features, using both PR-SZZ labels and manual ground truth for evaluation. This analysis reveals whether observed improvements are robust to label noise or if they are partly driven by overfitting to dataset noise.

By focusing on PR-SZZ, we do not claim that our results can be directly generalized to all SZZ variants. Nevertheless, all SZZ variants share the same general principle of

Table 1 Population counts $N_{\text{project,stratum}}$ per project and model×PR-SZZ stratum.

Project	S1	S2	S3	S4	Sum
`Airflow`	221	446	72	798	**1,537**
`Angular`	3	435	4	1,489	**1,931**
`Calcite`	10	23	6	80	**119**
`Jenkins`	16	98	6	497	**617**
`Kafka`	78	235	35	473	**821**
`Pulsar`	248	307	99	481	**1,135**
Sum	**576**	**1,544**	**222**	**3,818**	**6,160**

tracing bug-fixing changes back to earlier candidate bug-inducing changes. We therefore argue that the mechanisms studied in this work are not specific to PR-SZZ alone, and that the general findings are relevant beyond this particular variant. We discuss this limitation and its implications in more detail in Section 6.2.

4.2 Population and Data Collection

We train models using the published defect datasets and subsequently evaluate them on a disjoint evaluation period from 1 January 2021 to 30 June 2021 (2021-H1). The evaluation population consists of all commits created during this period across the six study projects. This yields a total of $N = 6{,}160$ commits. For each commit c in this population we extracted the following information:

- Project identifier and commit metadata (commit hash, author, timestamp, message).
- A binary PR-SZZ label $\widehat{y}^{\text{prszz}}(c) \in \{0, 1\}$ by running PR-SZZ on the 2021-H1 period, indicating whether PR-SZZ identifies c as defect-inducing.
- The full feature vector used in [1] (traditional, process and code features), computed and normalized in the same way as for the training period.
- A continuous defect-prediction score $\hat{p}(c)$ for every commit in 2021-H1 produced by the model trained on the training dataset from [1].
- A binary model prediction $\widehat{y}^{\text{model}}(c) \in \{0, 1\}$ obtained by thresholding $\hat{p}(c)$ at the decision threshold, which was fixed at 0.5.

The combination of $\widehat{y}^{\text{model}}(c)$ and $\widehat{y}^{\text{prszz}}(c)$ partitions the 6,160 commits into four strata, defined as:

$$\begin{aligned}
&\text{S1} : \widehat{y}^{\text{model}} = 1 \ \wedge \ \widehat{y}^{\text{prszz}} = 1 && \text{(model positive, PR-SZZ positive)} \\
&\text{S2} : \widehat{y}^{\text{model}} = 1 \ \wedge \ \widehat{y}^{\text{prszz}} = 0 && \text{(model positive, PR-SZZ negative)} \\
&\text{S3} : \widehat{y}^{\text{model}} = 0 \ \wedge \ \widehat{y}^{\text{prszz}} = 1 && \text{(model negative, PR-SZZ positive)} \\
&\text{S4} : \widehat{y}^{\text{model}} = 0 \ \wedge \ \widehat{y}^{\text{prszz}} = 0 && \text{(model negative, PR-SZZ negative).}
\end{aligned}$$

The stratum sizes per project and globally are displayed in Table 1.

The strata correspond to approximately 9.4% (S1), 25.1% (S2), 3.6% (S3), and 62.0% (S4) of all commits. This distribution shows a strong class and stratum imbalance: only about 34% of commits are predicted positive by the model (S1+S2), only about 13% are labeled positive by PR-SZZ (S1+S3), and the disagreement strata S2 and S3 together account for roughly 29% of the population, with the rarest and most critical disagreement stratum S3 (model negative, PR-SZZ positive) containing just 3.6% of all commits. The six projects under investigation differ substantially in activity, from 119 commits in `Calcite` to 1,931 commits in `Angular`. Also strata population varies across projects, with a very limited number of true positives recognized by the SZZ algorithm in the case for `Angular`, `Calcite`, and `Jenkins`.

4.3 Evaluation Design

We employ two complementary evaluation strategies using one defect prediction model per project trained with data from the published defect datasets, strictly before the evaluation period. The remainder of this study refers to the model under investigation just as model.

Automatic evaluation on the full population.

In the first step, we perform a fully automatic evaluation on all 6,160 commits by treating the PR-SZZ labels $\widehat{y}^{\text{prszz}}$ as ground truth. This yields standard performance measures (precision, recall, F1, AUC) under the same assumptions as the majority of prior JIT-DP studies. This evaluation is computationally inexpensive and provides complete coverage of the population, but its validity is limited by the residual noise in PR-SZZ labels.

Manual evaluation on a stratified sample.

In the second step, we construct a manually curated validation dataset sampled from the same population (see Section 4.3.1). For these sampled commits, we determine defect-inducing status by manual inspection (see Section 4.3.3) and treat the resulting labels as ground truth. We then re-evaluate the model against this manual reference, using appropriate sampling weights which allows us to obtain approximately unbiased estimates under the usual assumptions of stratified sampling. This evaluation is costly but provides higher internal validity and allows us to quantify the extent to which PR-SZZ misclassifications inflate or distort reported performance compared to human labeling.

Both automatic and manual evaluations use the same trained model and decision threshold; only the assumed ground truth (PR-SZZ vs. manual labels) differs.

4.3.1 Sampling Strategy

For the manual evaluation of the model performance on a validated dataset and the assessment whether misclassifications of SZZ commits actually lead to a change in reported model performance we construct a manually validated dataset. In order to obtain this dataset with manageable effort we draw a sample of the 2021-H1 commits that represent a statistically meaningful subset of the whole distribution.

A central methodological choice is whether to draw a simple random sample of commits or to stratify by model outcome and PR-SZZ label. In principle, a simple random sample yields unbiased estimates of overall precision and recall without further adjustment. In our setting, however, random sampling is statistically inefficient and makes poor use of limited manual labeling capacity. Given the population distribution (Section 4.2), a simple random sample of 400 commits would contain approximately 62% commits from S4, 9% from S1, and only about 29% from the disagreement strata S2 and S3 combined. Because most S4 commits are easy negatives, where both the model and PR-SZZ agree that the commit is clean, they contribute little information about error mechanisms and inflate estimator variance relative to their labeling cost. In contrast, the most consequential cases for real-world deployment, false positives (S2) and false negatives (S3), would be sparsely sampled, resulting in high-variance estimates of precision and recall and limited power to analyze systematic error patterns. The agreement-positive stratum (S1) and the disagreement stratum (S3) are particularly sparse, especially in `Angular` and `Jenkins`, where they comprise only 7 and 22 commits, respectively. This pronounced imbalance, both across projects and across strata, motivates our use of a stratified sampling design with deliberate oversampling of rare but informative strata. Stratification reallocates labeling effort toward commits with higher expected information content, substantially reducing estimator variance under a constrained labeling budget.

We adopt a two-stage stratified sampling design [3] with a total labeling budget of n equaling roughly 400 commits (approximately 6% of all commits), which should give statistically significant results. First, we allocate the total sample size across projects approximately proportional to each project's share of the 2021-H1 commits, subject to a minimum of 40 labeled commits (10% of the sampling budget) per project to ensure representation even for small projects. This guarantees that large projects such as `Angular` and `Airflow` are weighted according to their importance in the population, while smaller projects like `Calcite` and `Jenkins` still receive enough labeled examples to support project-level analysis. Within each project, we stratify commits into the four model×PR-SZZ strata S1–S4. We target an approximately equal allocation across the four strata within each project (Equal Stratified Sampling). If a project-stratum cell $(\text{project}, \text{stratum})$ contains fewer commits than the planned allocation, we sample all commits from that cell and redistribute the remaining sample size to other strata within the same project. This guarantees that:

- all projects are represented in the validation dataset;
- rare but critical cases are deliberately over-sampled; and
- each project has labeled examples from all strata, enabling the analysis of project-specific error patterns.

Table 2 reports the resulting strata and corresponding sample sizes per project. Overall, we manually label approximately 7% of all commits, with project-level coverage ranging from 6.1% (`Kafka`) to 33.6% (`Calcite`) and a median of 6.3%. We intentionally oversample `Calcite` and `Jenkins` to 40 commits each, yielding a total labeled dataset of 420 commits.

Table 2 Allocated sample sizes $n_{\text{project,stratum}}$ per project and model×PR-SZZ stratum (2021-H1).

Project	S1	S2	S3	S4	Sum
`Airflow`	25	24	24	24	97
`Angular`	3	58	4	57	122
`Calcite`	10	12	6	12	40
`Jenkins`	11	11	6	12	40
`Kafka`	13	13	12	12	50
`Pulsar`	18	18	18	17	71
Sum	**80**	**136**	**70**	**134**	**420**

4.3.2 Weighting and Population-level Analysis

Because the sampling design oversamples certain strata and projects relative to their prevalence in the underlying population, naive (unweighted) estimates would not reflect population-level performance. We therefore use inverse-probability weighting to obtain unbiased estimates [16]. This corresponds to the Horvitz–Thompson estimator [12], which provides unbiased estimates of population totals under stratified sampling. Let $N_{p,s}$ denote the number of commits in project p and stratum s in the full population (Table 1), and let $n_{p,s}$ denote the number of sampled commits in that project-stratum cell. Each sampled commit i in (p, s) is assigned weight

$$w_{p,s} = \frac{N_{p,s}}{n_{p,s}}.$$

This weight then allows us to estimate the number of commits in each strata for the full 2021-H1 population of a manually evaluated dataset. Precision, recall, and other performance metrics are computed as ratio estimators derived from these weighted totals [16].

To quantify the uncertainty of our estimates, we apply a stratified bootstrap with 2,000 replicates [6]. In each replicate, we resample with replacement within each project-stratum cell (p, s), preserving the original sampled cell sizes $n_{p,s}$, recompute all weighted performance measures, and derive 95% confidence intervals from the empirical bootstrap distribution (2.5^{th} and 97.5^{th} percentiles).

4.3.3 Manual Labeling Procedure

For each sampled commit, we manually determine whether it is truly defect-inducing by analyzing the project history. To support this process, we also run three Large Language Models (LLM) chat-completion tasks (OpenAI GPT-5) for each commit, using three different templated prompts. Each prompt asks the LLM to classify the commit as defect-inducing or not, based on the git diff as well as the related issue and pull-request content. The LLM responses are not directly used to determine the label but mark cases where additional scrutiny is needed. The manual procedure therefore combines human judgment with LLM-based assistance, taking advantage of the strengths of both. However, the final decision is always made by the human reviewer and is never

delegated to the LLM. Overall, this procedure is intended to be more comprehensive and context-aware than the automated SZZ algorithm, which relies only on commit history and code changes.

While the manual procedure follows the general logic of SZZ (tracing commits and looking for defect/issue reports), it starts from the potentially bug-inducing commits and looks at changed lines, as well as referenced issues, comments, and pull requests directly. We employ the GitHub Search to search for commit hashes and issue/pull-request references in later commits. The process hence uses additional contextual information (issue discussions, pull requests, commit rationale) that is not fully available to automated SZZ variants and uses human judgment in those extra cases. Further, for each commit we review changes and look for introduced syntactic or semantic issues. This adds additional information to the process and reduces spurious links that PR-SZZ might introduce.

A commit is labeled defect-inducing if it introduces or modifies code that is later changed by a bug-fixing commit addressing a functional defect (e.g., crash, incorrect behavior, data loss) or clearly referred to as the introducing commit by linkage or collaborator discussions; performance optimizations or purely cosmetic issues are excluded. Otherwise, the commit is labeled as clean. Each sampled commit is assessed by a human reviewer with experience in software engineering and familiarity with the studied projects. In detail the analysis for each commit looks like this:

- For a candidate commit, one human reviewer inspect the commit message, code diff, and the set of modified files. They check for any distinctive feature that might indicate a defect.
- They examine associated issue tracker entries (if any), pull requests, and follow-up bug-fixing commits referencing the same files or issues.
- They look at the current state of changed files and investigate if any changes to these files in the meantime point towards a bug fix related to the candidate commit.
- They check the results of SZZ and double-check if a found link is actually correct or introduced by a spurious link to a non-related fixing issue.
- In special cases where no bug-fixing commit is found but the reviewer suspect a introduced defect, they raise the issue in the issue tracker of the project and wait for comment from the maintainers (this was the case for 1 out of the 420 samples: the bug was acknowledged by the maintainers and subsequentially fixed).
- A first decision is made based on the available information.
- They utilize LLMs to review the commit and check their answers to have more context for the decisions. If reviewer and LLM are in agreement, the decision is straightforward. If they are in disagreement, the commits are added to a list of edge cases.

The identified edge-cases where multiple LLM responses and the human reviewer decision are in disagreement (48 cases; 11.4%) are discussed in a consensus meeting with a second human reviewer, during which reviewers revisit the relevant evidence

(diffs, issues, subsequent fixes) and decide these edge-cases. The outcome of this process is a manually labeled validation dataset of 420 commits, each with a binary ground-truth label $y^{\text{manual}}(c) \in \{0, 1\}$ that is used as ground truth for the subsequent study.

5 Results

In this section, we report the results of our study. We first present descriptive statistics for the population and sample, then answer posed research questions.

5.1 Descriptive Statistics of Population and Sample

The investigated 2021-H1 evaluation population comprises $N = 6{,}160$ commits across the six projects. PR-SZZ labels 12.9% of these commits as defect-inducing, while the model predicts 34.4% as defect-inducing (Section 4.2). From this population, we manually labeled $n = 420$ commits (6.8% of all commits), selected according to the stratified sampling design described in Section 4.3.1. The number of commits resulted from our basis of 400 commits plus the mandated oversampling of too dimly populated strata. As intended, the sample is enriched for high-risk and disagreement cases between the model and PR-SZZ, and therefore does not mirror the population prevalence of labels when viewed unweighted.

Table 3 Confusion matrix of PR-SZZ versus manual ground truth in the stratified sample (420 commits, unweighted).

	Manual ground truth	
PR-SZZ label	Defect-inducing	Clean
Defect-inducing	57 (TP)	93 (FP)
Clean	17 (FN)	253 (TN)

Table 4 Confusion matrix of PR-SZZ versus manual ground truth in the population estimate (weighted).

	Manual ground truth	
PR-SZZ label	Defect-inducing	Clean
Defect-inducing	309.8 (TP)	488.2 (FP)
Clean	296.6 (FN)	5056.4 (TN)

Table 3 shows the unweighted confusion matrix between PR-SZZ and manual ground truth for the 420 manually validated commits. In this sample, PR-SZZ marks 150 commits as defect-inducing (35.7%). PR-SZZ therefore mislabels 26.2% of commits in the raw sample, mainly due to false positives. This discrepancy is expected because the sample deliberately oversamples rare but informative strata (S1 and S3);

it is not a sampling error and is corrected in all subsequent population-level estimates using inverse-probability weighting (Section 4) to project results back to the full 2021-H1 population. Weighted estimates for the full population, as shown in Table 4, suggest that PR-SZZ actually misclassified about 12.7% of commits. In the remainder of this study we refer to the weighted estimates as the manual ground truth for further evaluations.

5.2 RQ1: PR-SZZ Label Noise

In this section, we evaluate RQ1 and assess to what extent PR-SZZ mislabels commits. We aim to understand if there is a clear directional bias in the SZZ results and how this affects the investigated projects. Building on the descriptive statistics in Section 5.1, we compare labels derived by PR-SZZ for the manually sampled commits to these manually validated labels. We first analyze overall label quality at the population level, then examine variation across projects and strata.

Weighted up to the full 2021-H1 population (N = 6,160), PR-SZZ exhibits the following performance relative to manual labels:

- **Misclassification rate:** 12.7% of commits are misclassified (either false positives or false negatives).
- **Precision:** About 38.8% of commits labeled as defect-inducing by PR-SZZ are truly defect-inducing.
- **Recall:** PR-SZZ identifies roughly 51.1% of all truly defect-inducing commits.

In terms of prevalence, the true fraction of defect-inducing commits in the population is approximately 9.8%, whereas PR-SZZ marks about 12.9% of commits as defect-inducing. Thus, PR-SZZ overestimates the proportion of defect-inducing commits by roughly 24% relative to manual ground truth. Decomposing the misclassification rate, about 7.9% of all commits are false-positives (clean commits labeled as defect-inducing) and 4.8% are false-negatives (defect-inducing commits labeled as clean). These descriptive results indicate directional label noise: PR-SZZ overestimates the number of defect-inducing commits rather than making purely random errors. The label noise is skewed towards false positives.

Table 5 Assigned PR-SZZ versus manual ground truth labels for each project. Defective and Clean columns show the percentage of commits that are labeled as such.

Project	PR-SZZ		Manual		Misclassification rate (%)
	Defective	Clean	Defective	Clean	
Airflow	19.1	80.9	7.1	92.9	14.3
Angular	0.4	99.6	3.5	96.5	3.5
Calcite	13.5	86.5	5.0	95.0	8.4
Jenkins	3.6	96.4	11.1	88.9	11.7
Kafka	13.8	86.2	11.3	88.7	11.3
Pulsar	30.6	69.4	23.1	76.9	28.4

Overall, PR-SZZ behaves as noticeably noisy defect labeler: it catches slightly more than half of all true defect-inducing commits, and fewer than half of its positive labels correspond to real defect introductions. PR-SZZ accuracy is not uniform across the six projects. Table 5 summarizes, for each project, the weighted misclassification rate and the prevalence of defect-inducing commits under manual labels and under PR-SZZ. Several patterns emerge:

- In `Airflow`, `Calcite`, `Kafka`, and especially `Pulsar`, PR-SZZ marks too many defect-inducing commits. For example, in `Airflow` the true defect-inducing rate is 7.1%, whereas PR-SZZ marks 19.1% of commits as defect-inducing; in `Kafka`, the difference is less pronounced with 11.3% versus 13.8%.
- In `Angular` and `Jenkins`, PR-SZZ is highly conservative: it detects far fewer defect-inducing commits than the manual ground truth (0.4% vs. 3.5%, and 3.6% vs. 11.1%, respectively). In these projects, most PR-SZZ errors stem from false negatives rather than false positives; recall is low, and many true defect-inducing commits are not contained in the produced dataset.

These results indicate that the nature and magnitude of PR-SZZ noise depends on project context. In some systems PR-SZZ predominantly overestimates defect-inducing activity, while in others it underestimates it. Any evaluation that treats PR-SZZ labels as ground truth therefore inherits project-specific biases. These biases are likely due to the way developers track defects in the respective projects.

To better understand where PR-SZZ fails, we examine misclassification rates within the four strata (Section 4.2):

- S1 (model=1, PR-SZZ=1): misclassification rate 59.3% (weighted $N \approx 576$).
- S2 (model=1, PR-SZZ=0): misclassification rate 9.3% (weighted $N \approx 1{,}544$).
- S3 (model=0, PR-SZZ=1): misclassification rate 66.0% (weighted $N \approx 222$).
- S4 (model=0, PR-SZZ=0): misclassification rate 4.0% (weighted $N \approx 3{,}818$).

Across the four strata, PR-SZZ's errors are highly concentrated in its positive labels. When PR-SZZ marks a commit as defect-inducing (S1 and S3), it is wrong more often than not, with misclassification rates of 59% and 66%, respectively. By contrast, its negative labels are far more reliable: disagreement with the model still yields a low error rate (9% in S2), and joint negative agreement (S4) is the most accurate region of all, with only 4% misclassified. Thus, PR-SZZ is generally more trustworthy when it labels commits as clean but substantially less reliable when it labels them as defect-inducing.

These patterns have two important implications. First, they validate our stratified sampling design: by oversampling the disagreement strata and then reweighting, we obtain unbiased population-level estimates despite the skewed distribution of errors. Second, they highlight how PR-SZZ label noise can interact with model evaluation. When PR-SZZ is wrong but the model agrees with its label, both will appear correct under PR-SZZ-based evaluation despite being wrong relative to the manual ground truth. This dynamic, examined in detail in RQ2, helps explain why models can seem

to perform well under PR-SZZ labels even when their true defect-prediction capability is more limited.

In summary, PR-SZZ substantially reduces label noise compared to classical SZZ [2], but it remains an imperfect proxy. At the population level, PR-SZZ misclassified an estimated 12.7% of commits: only about 39% of its positive labels are correct, and it recovers roughly 51% of truly defect-inducing commits. On average, this translates into an inflation of defect-inducing prevalence from 9.8% (manual ground truth) to 12.9% (PR-SZZ). However, this misclassification bias is not uniform. Across projects, PR-SZZ's error rates and prevalence distortions vary widely: some projects show pronounced overestimation of defective commits, whereas others exhibit an underestimation with many missed defect-inducing commits. These project-specific differences imply that evaluations treating PR-SZZ as ground truth will inherit non-random, context-dependent biases rather than a single, stable noise profile. This shows again how reliant the two-phased SZZ algorithm is on the project internal structures and workflows to label and triage defects. Finally, the stratified analysis reveals that PR-SZZ errors are concentrated in its positive labels. This structure matters for downstream evaluation because it can mask model errors or amplify apparent model performance depending on whether the model agrees with PR-SZZ in the high-noise regions. In the next section, we evaluate if this asymmetric, project-dependent noise can materially bias standard model evaluation conclusions.

5.3 RQ2: Evaluation Bias

In this section, we first report the model's performance on the full 2021-H1 population when evaluated against PR-SZZ labels (*automatic evaluation*), then re-estimate the same metrics against manual ground truth using the stratified, weighted design described in Section 4.3.1. We then analyze per-project differences and examine how PR-SZZ label noise interacts with the model to inflate reported (*manual evaluation*).

Table 6 summaries the model performance on all 6,160 commits under the two evaluation regimes. Consequently, Table 7 breaks up the model performance per project.

Table 6 Overall model performance on 2021-H1 commits under PR-SZZ labels (automatic evaluation) and manual ground truth (weighted population estimates) averaged across all projects. Bootstrap 95% CIs are shown for manual evaluation.

Metric	**PR-SZZ ground truth**	**Manual ground truth**
Accuracy	0.71	0.68 (0.65–0.72)
Precision	0.27	0.18 (0.13–0.24)
Recall	0.72	0.62 (0.47–0.79)
F1-score	0.39	0.28 (0.20–0.35)
AUC	0.80	0.75

Manual positive prevalence $\approx$ 9.8%; PR-SZZ positive prevalence $\approx$ 12.9%.

Table 7 Per-project model performance under PR-SZZ labels and manual ground truth. Manual metrics are weighted population estimates; PR-SZZ metrics are computed on the full 2021-H1 population.

	PR-SZZ ground truth			Manual ground truth		
Project	P	R	F1	P	R	F1
`Airflow`	0.33	0.75	0.46	0.13	0.81	0.23
`Angular`	0.01	0.43	0.01	0.09	0.58	0.15
`Calcite`	0.30	0.63	0.41	0.09	0.50	0.15
`Jenkins`	0.14	0.73	0.24	0.22	0.37	0.28
`Kafka`	0.25	0.69	0.37	0.25	0.84	0.39
`Pulsar`	0.45	0.71	0.55	0.26	0.55	0.35

5.3.1 Automatic Evaluation Regime

Under the standard evaluation (PR-SZZ as ground truth), the model appears to perform reasonably well and broadly mirrors the cross-validation results reported in the Bludau et al. study[1]: recall is 0.72 and precision 0.27, with F1 = 0.39 and AUC = 0.80. The model produces many false positives in this low-prevalence setting, which leads to lower precision and therefore F1.

At the project level, a similar picture emerges: projects that were easier to model during cross-validation (e.g., Pulsar, Airflow, Calcite) also tend to retain better performance in the unseen 2021-H1 period, while Jenkins and especially Angular remain challenging. However, the most notable discrepancy is a particular drop in precision for some projects. The extreme case is Angular (precision = 0.01, recall = 0.43, F1 = 0.01), where a very small number of PR-SZZ positives combined with a large number of predicted positives leads to hundreds of false positives and a near-zero precision. Such shifts may reflect limitations or noise in PR-SZZ labeling itself—meaning apparent "errors" can arise both from genuine model drift and from imperfect ground truth. Moreover when defect tracking methods change over time this will affect the outcome of SZZ and likely lead to skewed model evaluations.

In high-stakes scenarios, recall is often considered the primary objective because missing a defect-inducing commit is costlier than flagging an additional clean one. In our evaluation we find that recall and ranking performance (AUC) is less sensitive to label noise. This suggests that models may still be useful for prioritization and triage. This view particularly leans towards effort-aware evaluation settings [13].

5.3.2 Manual Evaluation Regime

When evaluated against manual ground truth (Table 6), on average all measured performance metrics deteriorate. Figure 1 visualizes the discrepancy. Precision drops to 0.18 and recall to 0.62, yielding F1 = 0.28. Bootstrap analysis shows that these differences are statistically significant: the 95% confidence interval for the inflation in precision is [0.03, 0.14], and for F1 is [0.04, 0.19], both excluding zero. In practical terms, this means that:

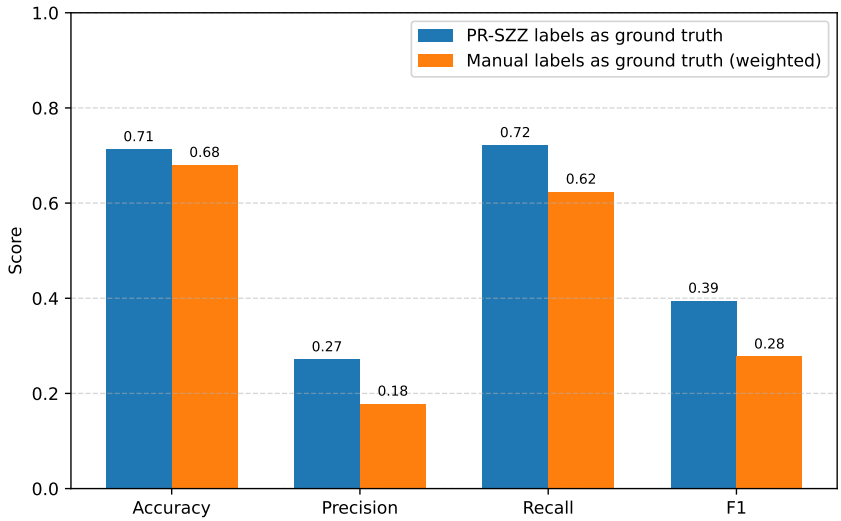


Fig. 1 Model performance difference under PR-SZZ labels (blue bars) and manual ground truth (orange bars) across all projects.

- Only about one in five commits flagged by the model is truly defect-inducing; the remaining four are false alarms.
- The model detects roughly 62% of truly defect-inducing commits, missing about one third.

The impact of PR-SZZ labels on reported performance is clearest for precision and F1. This confirms that even a comparatively refined SZZ variant such as PR-SZZ induces non-trivial optimism in reported performance. Using the stratified bootstrap, we estimate that PR-SZZ-based evaluation inflates:

- **Precision** by +0.09 absolute (95% CI: 0.03–0.14), corresponding to a relative increase of approximately 34%.
- **F1-score** by +0.12 absolute (95% CI: 0.04–0.19), corresponding to a relative increase of roughly 30%.

For recall and accuracy, the point estimates under PR-SZZ (0.71 and 0.72) are also higher than the manual estimates (0.68 and 0.62), but the bootstrap confidence intervals for the differences include zero; we therefore treat the evidence for recall and accuracy inflation as suggestive rather than conclusive. Ranking performance is comparatively robust: AUC decreases only slightly, from 0.80 under PR-SZZ labels to 0.75 under manual labels. This indicates that while PR-SZZ noise substantially distorts

threshold-dependent measures such as precision and F1, it has a smaller impact on the model's ability to rank commits.

At the project level (Table 7), the direction and magnitude of this inflation vary:

- Precision remains low in all projects. Under manual evaluation, precision ranges only from 0.09 to 0.26. In other words, between 74% and 91% of flagged commits are false alarms. This pattern is consistent with the global precision of 0.18 and suggests that the model is best suited for triage and prioritization rather than as a strict gatekeeper.
- In `Airflow`, `Calcite`, and `Pulsar`, PR-SZZ-based evaluation substantially overestimates precision and F1. For example, in `Pulsar`, precision increases from 0.26 (manual) to 0.45 (PR-SZZ) and F1 from 0.35 to 0.55. These are all projects where the dataset evaluation showed a high overestimation of defect-inducing commits in PR-SZZ.
- In `Angular`, PR-SZZ labels produce extremely low apparent performance (precision $\approx$ 0.01, F1 $\approx$ 0.01), because PR-SZZ almost never labels commits as defect-inducing in this project. Here, the main effect of PR-SZZ is to underestimate both prevalence and model performance.
- In `Jenkins`, PR-SZZ-based evaluation increases recall sharply (0.37 to 0.73) but lowers precision (0.22 to 0.14), leading to a small net decrease in F1.
- In four of the six projects (`Kafka` (0.84), `Airflow` (0.81), `Angular` (0.58), `Pulsar` (0.55)) the model attains recall above 0.50 when evaluated against manual truth. This means that, in these systems, the model still identifies the majority of truly defect-inducing commits.

In conclusion, these results show that PR-SZZ misclassifications do not introduce a simple uniform bias. Instead, they produce heterogeneous distortions that depend on the project and on the local interaction between PR-SZZ and the model, with the aggregate effect of inflating precision and F1 when pooling across projects. These differences reflect underlying variation in codebase, process, and labeling quality, and caution against assuming uniform performance across projects.

5.3.3 Interaction Between PR-SZZ Errors and Model Predictions

To understand why PR-SZZ labels inflate model performance, we analyze how PR-SZZ errors and model predictions align at the commit level. Using the weighted population estimates, we partition the 2021-H1 commits into four categories, shown in Figure 2.

Among the PR-SZZ mislabeled commits (bottom row), the model is more likely to agree with PR-SZZ's incorrect label (7.6% of all commits) than to correct it (4.7%). Under PR-SZZ-based evaluation, these cases are counted as correct predictions, even though both PR-SZZ and the model disagree with the manual ground truth. This systematically inflates precision and F1. In other words, on the commits where PR-SZZ is wrong, the model appears to agree with the PR-SZZ labels.

To illustrate how this interaction affects standard performance measures, Figure 3 contrasts the same model predictions when evaluated against PR-SZZ labels versus manual labels.

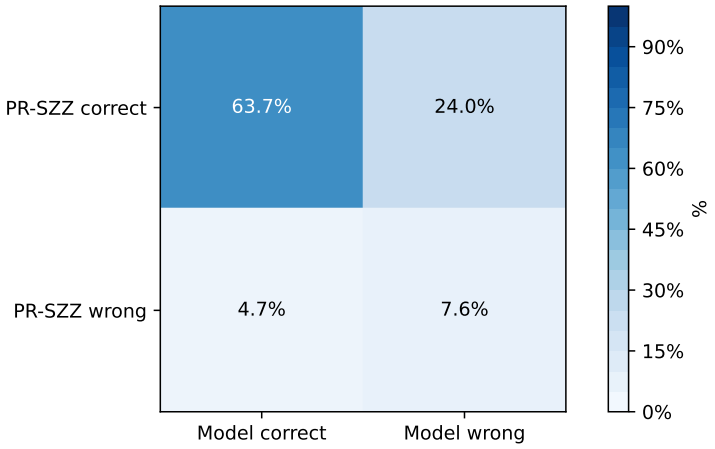


Fig. 2 Label correctness of PR-SZZ (y-axis) and model prediction correctness (x-axis) according to manual ground truth (in % of commits).

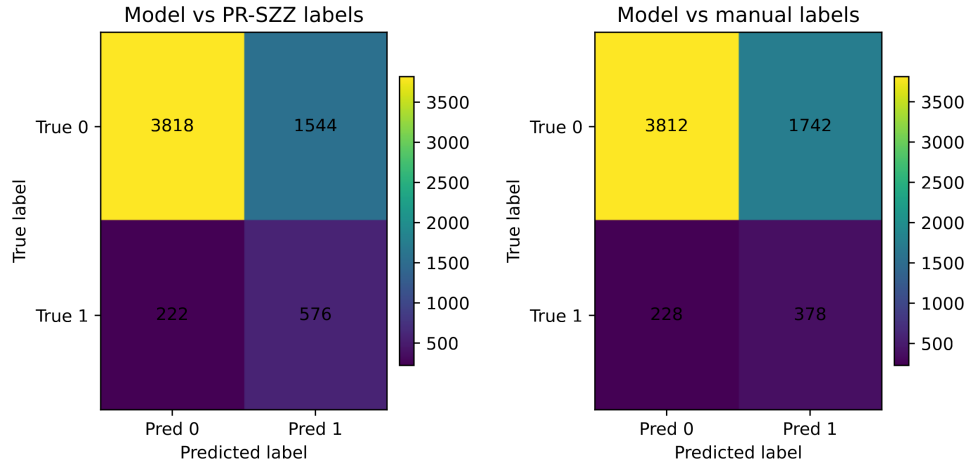


Fig. 3 Interaction between PR-SZZ labeling errors and model predictions: evaluation against PR-SZZ (left) versus manual labels (right) for 2021-H1 commits.

The key difference is concentrated in the $Pred = 1$ region: PR-SZZ credits the model with 576 true positives, whereas manual labeling confirms only 378, i.e., 198 commits are counted as positives by PR-SZZ but are not positive under manual ground truth. Symmetrically, the number of false positives among predicted positives increases from 1,544 (PR-SZZ) to 1,742 (manual). In contrast, the Pred=0 region is nearly unchanged (false negatives: 222 vs. 228; true negatives: 3,818 vs. 3,812), indicating

that the inflation is not driven by additional missed defects but by PR-SZZ reclassifying a substantial portion of the model's flagged commits as "true" bug-inducing. On average, this produces a clear upward bias in performance estimates, demonstrating that PR-SZZ errors are not simply independent noise but interact with the model's decision boundary: these mislabels disproportionately occur in the subset of commits the model predicts as defect-inducing, thereby overstating real-world usefulness. However, as noted earlier, this pattern does not hold for every project in the evaluation and therefore should not be treated as a universal rule.

5.4 RQ3: Effect on Existing Research

To answer RQ3, we reassess whether the significant model performance improvements attributed to the additional features proposed in [1] also hold when compared to the manually labeled dataset, using the stratified weighting and bootstrap procedures described in Section 4.3.1. Specifically, we compare a baseline model M_0, which uses the original feature set (Baseline), with an enhanced model M_1, which combines the original features with the newly proposed semantic features (Enhanced).

Table 8 shows that under manual ground truth the enhanced model does not outperform the baseline. The baseline achieves precision 0.21 and recall 0.65. The enhanced model attains precision 0.18 and recall 0.62. Ranking performance is less affected (AUC 0.75 vs. 0.76).

Table 8 Baseline vs. enhanced model performance under manual ground truth (weighted population estimates, with 95% bootstrap confidence intervals).

Metric	**Baseline** M_0		**Enhanced** M_1	
Precision	0.21	[0.15, 0.27]	0.18	[0.13, 0.23]
Recall	0.65	[0.50, 0.81]	0.62	[0.48, 0.78]
F1-score	0.31	[0.24, 0.40]	0.28	[0.20, 0.35]
AUC	0.75	[0.64, 0.85]	0.75	[0.66, 0.84]

Bootstrap deltas confirm that the degradation is concentrated in threshold-dependent metrics (Table 9). Precision decreases by about -0.03 and F1 by about -0.04, while differences in recall and AUC include zero, suggesting no significant difference between the models in these dimensions. Practically, adding semantic features increases false alarms without improving defect coverage.

This reversal relative to PR-SZZ-based results is consistent with the broader finding of this study: models trained and evaluated on PR-SZZ can be rewarded for fitting PR-SZZ's labeling behavior, rather than the underlying phenomenon of defect introduction. Additional semantic features increase model expressiveness, which can improve imitation of the PR-SZZ labeling function under PR-SZZ evaluation, yet translate into worse precision under manual ground truth.

Crucially, this should not be read as evidence that semantic features are intrinsically unhelpful. Rather, it indicates that in the standard SZZ-based training and evaluation pipeline, apparent gains can be fragile and may reverse when assessed

Table 9 Differences in performance between enhanced and baseline models under manual ground truth (enhanced − baseline, with 95% bootstrap confidence intervals).

Metric	Δ mean	CI low	CI high
Precision	−0.03	−0.06	−0.00
Recall	−0.02	−0.08	0.04
F1-score	−0.04	−0.07	−0.00
AUC	+0.01	−0.03	0.05

against higher-quality ground truth. Two mechanisms can explain this. First, expanding the feature space increases variance and can induce overfitting, so the model may learn distinctive characteristics of the training period or project context that do not generalize to the validation period distribution. Second, because training relies on noisy PR-SZZ labels, added capacity can be used to learn the noise: semantic features may align with systematic mislabeling, making the model better at predicting PR-SZZ positives than true positives. Thus, representational power that could capture defect-proneness can instead capture artifacts of the labeling process. Interestingly, some of the reasons why SZZ misclassifies commits align well with features from defect prediction research. For instance, the size of a commit correlates with the probability of SZZ flagging it as defect inducing. At the same time defect prediction research shows that the size of a commit is the most important feature in defect prediction models [13, 1]. This distinction is interpretively central: improvements under PR-SZZ do not necessarily translate to improved real-world defect detection; they may simply reflect closer approximation to the SZZ labeler. Therefore, feature advances should be validated against higher-quality ground truth or noise-robust protocols to avoid conflating better imitation of SZZ-derived labels with genuine progress in defect prediction. Our qualitative analysis in Section 6, still suggests benefits of semantic features for specific defect categories (e.g., refactoring-induced and distributed-system bugs), but such gains are not reliably captured by commit-level metrics on the limited manually labeled sample. Overall, the comparison reinforces the need for stronger ground truth and evaluation designs that are robust to label noise when making incremental claims about feature engineering or model improvements.

6 Discussion

This study set out to quantify how accurate PR-SZZ labels are relative to manual ground truth (RQ1), how PR-SZZ misclassifications distort commonly reported model metrics (RQ2) and how this affects previous results (RQ3). Across 6,160 commits from January–June 2021 and a stratified manually validated subset of 420 commits, three findings stand out: PR-SZZ introduces substantial label noise; model performance degrades when evaluated against manual ground truth; and previous results of JIT-DP study need to be assessed with care and are only valid under specific assumptions about the dataset.

In comparison to our manual validated sample, PR-SZZ mislabels about 12.7% of all commits, with a bias towards false positives. It overestimates the prevalence of defect-inducing commits (12.9% vs. 9.8%), and its positive labels are only moderately reliable. Project-level results show that PR-SZZ's error profile varies substantially. This heterogeneity cautions against transferring claims about PR-SZZ quality from one ecosystem or time period to another, especially when development processes or issue labeling conventions change. These PR-SZZ misclassifications have a clear and statistically significant impact on model evaluation. When PR-SZZ labels are treated as ground truth, the model precision increases from 0.18 to 0.27 and its F1-score from 0.28 to 0.39 on average. Bootstrap analysis indicates that these differences are unlikely to be due to sampling variability: PR-SZZ-based evaluation inflates precision by roughly 34% and F1 by about 30% relative to manual evaluation. By contrast, recall (14%) and ranking performance (AUC, 6%) is relatively robust, decreasing more slightly under manual labels and is not deemed statistically significant.

With a true defect-inducing prevalence of roughly 9.8%, a precision of 0.18 implies that around four out of five flagged commits are false alarms, creating a substantial screening burden. Although recall is comparatively less affected and the model can still surface many true defect introductions this benefit comes with a high false-positive rate that limits the model's usefulness as a stand-alone decision mechanism. This asymmetry is consistent with SZZ-style label noise: false positives dominate, disproportionately depressing precision while leaving recall less severely impacted. Accordingly, prior studies whose conclusions rely primarily on recall may be less affected than those emphasizing precision; however, their reported precision and implied developer-facing workload are likely optimistic and should be interpreted cautiously. In practice, these results support using models as a triage aid rather than as a hard gate or automated quality barrier, particularly given evidence that sustained false alarms can undermine developer trust and adoption.

6.1 Qualitative Findings: A Taxonomy of Labeling and Prediction Failure Modes

The manual inspection provides qualitative evidence that helps explain the quantitative misclassification patterns observed in the previous section. Rather than treating PR-SZZ errors as homogeneous noise, our analysis shows that they arise from several distinct mechanisms. We therefore organize the findings into two groups: SZZ-induced label-construction errors (L), which affect the correctness of the generated labels, and downstream prediction and observability limitations (M), which affect the ability of models and datasets to capture defect-inducing behavior. Figure 4 summarizes the qualitative taxonomy derived from the manual inspection. The taxonomy separates errors introduced during SZZ-based label construction from broader prediction and observability limitations that affect the interpretation and deployment of JIT-DP models.

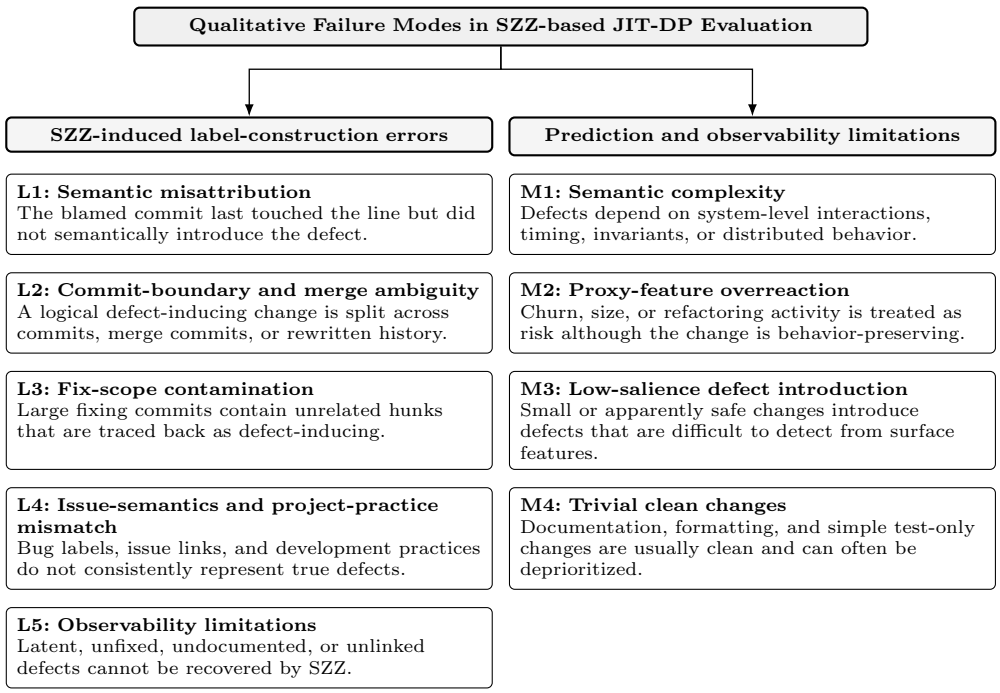


Fig. 4 Taxonomy of qualitative failure modes observed during manual validation. The taxonomy separates errors introduced during SZZ-based label construction from broader prediction and observability limitations.

6.1.1 SZZ-induced Label-Construction Errors

The qualitative analysis provides some insights in why SZZ variants may struggle with the identification of defect inducing commits in general.

L1: Semantic misattribution of defect causality.

The most direct error class occurs when PR-SZZ attributes a defect to the commit that last modified a blamed line, although the semantic cause of the defect lies in an earlier or different change. For example, commit `b49610e2` in `Jenkins` was marked as defect-inducing because it was implicated by a later bug-fixing commit. Manual inspection showed, however, that the fix `ebca2030` addressed a defect introduced by an earlier change rather than by `b49610e2` itself. This illustrates a fundamental limitation of line-based blame: the last commit touching a line is not necessarily the commit that introduced the faulty behavior. Such errors create false-positive SZZ labels and may also leave the actual inducing commits unlabeled.

L2: Commit-boundary and merge ambiguity.

A second class of errors arises when the same logical change is represented by multiple commits, merge commits, or rewritten history. In `Jenkins`, commits `23fbd2b9`

and `8ee48ec1` are semantically part of the same defect-inducing change, yet PR-SZZ and the model assign different labels to them. In such cases, the notion of a single defect-inducing commit becomes ambiguous. This error class is particularly relevant in projects with frequent merges, rebases, or squashed histories, where version-control structure does not map cleanly onto semantic change units.

L3: Fix-scope contamination.

PR-SZZ, and SZZ in general, struggles with large bug-fixing commits that touch many files or contain several unrelated hunks. When a fix includes formatting changes, documentation updates, test adjustments, refactorings, or cleanup work alongside the actual repair, SZZ may trace the entire affected context back to prior commits and mark semantically unrelated changes as defect-inducing. This produces inflated numbers of positive labels. The pattern is especially important because it directly explains why many PR-SZZ positives are false positives under manual validation.

L4: Issue-semantics and project-practice mismatch.

Some label errors are caused not by the blame algorithm alone, but by project-specific issue-tracking and development practices. In `Calcite`, for example, several issues labeled as bugs are closer to feature requests or enhancements, and large mixed-purpose commits often combine feature development, refactoring, and minor bug fixes. When such issues are used as inputs to SZZ, the resulting labels inherit this ambiguity. By contrast, during the investigated period, `Pulsar` contained more targeted changes and bug reports that corresponded more closely to genuine defects. This class of errors explains why PR-SZZ noise is project-dependent rather than uniform across repositories.

L5: Observability limitations.

Finally, SZZ-style datasets can only identify defects that eventually become observable through a linked fixing commit. Defects that remain latent, are fixed without an explicit bug link, are never reported, or are discovered only during manual inspection are absent from the generated ground truth. The `Angular` commit `ff45fbd0`, which introduced a parameter mismatch that had not yet been linked to a fixing commit, illustrates this limitation. This class differs from the previous ones: it is not merely a wrong attribution by PR-SZZ, but a boundary condition of any approach that equates defect-inducing changes with changes later linked to explicit fixes.

6.1.2 Downstream Prediction and Observability Limitations

The manual analysis also highlights the diversity and subtlety of defect-inducing changes, which places limitations on what current models can achieve.

M1: Semantic complexity.

Some defect-inducing commits are difficult to recognize because the defect only emerges from deep semantic interactions. For example, `Kafka` commit `d2cb2dc4` appears syntactically plausible, but the defect manifests through interactions between

distributed components, failure scenarios, and timing conditions. Such cases are unlikely to be captured by coarse process metrics, simple code features, or shallow static analysis alone. They expose a limitation of current prediction models rather than merely a limitation of SZZ.

M2: Proxy-feature overreaction.

A recurring modeling failure occurs when large or complex-looking changes are treated as risky even though they are behavior-preserving. For instance, `Pulsar` commit `306a448a` performs wide-ranging test adjustments but does not introduce a functional defect. Both SZZ-based labeling and learning-based models can overreact to such changes because size, churn, and file dispersion are often correlated with defect risk. This class is particularly important for evaluation bias: if SZZ also overlabels such commits as defect-inducing, models using churn-related features may appear more accurate under SZZ labels than under manual validation.

M3: Low-salience defect introduction.

The opposite problem occurs when small, apparently safe changes introduce real defects. In `Airflow`, commit `4e49adc0` introduced a subtle defect through a small refactoring that omitted necessary class behavior. Such changes may be missed by approaches that rely heavily on size, churn, or surface-level code characteristics. This class shows that defect risk is not simply proportional to change size and that small refactorings can require semantic inspection.

M4: Trivial clean changes.

Many commits in the stratum where both PR-SZZ and the model assign a clean label consist of documentation updates, formatting changes, or minor test changes. Manual inspection confirms that these commits are usually clean. This class is not an error mode in itself, but it is relevant for deployment: practical JIT defect prediction systems may be able to deprioritize clearly trivial changes, provided that such filtering remains conservative and acknowledges that rare small changes can still have large effects.

6.1.3 Implications of the Taxonomy

The taxonomy shows that PR-SZZ label noise is not random. Many errors arise from systematic mismatches between line-level blame, issue-tracker semantics, version-control structure, and the semantic notion of defect causality. This explains why the resulting noise is directional, project-dependent, and concentrated among commits labeled as defect-inducing. It also explains why models trained and evaluated on PR-SZZ labels may learn artifacts of the labeling process, particularly when their features correlate with churn, change size, or other signals used implicitly by SZZ.

At the same time, the taxonomy shows that improving JIT defect prediction requires more than refining SZZ heuristics. Some defects are latent, semantically subtle, or only observable through complex system behavior. Future datasets and models therefore need to combine more reliable ground-truth construction with complementary analysis techniques, including process metrics, static analysis, semantic code analysis, and manual validation.

6.2 Generalizability to Other SZZ Variants

In this study, we evaluated the impact of misclassifications produced by PR-SZZ, a refined SZZ variant, and their influence on model performance. However, many other SZZ variants exist, each relying on different heuristics and assumptions. As a result, the specific error profiles observed for PR-SZZ may not generalize directly to all other variants. Still, the broader conclusion that SZZ-based labels are noisy is likely to hold across variants, even if the magnitude and exact nature of the resulting distortions in model evaluation differ.

We argue that our findings generalize, at least in principle, for several reasons. First, PR-SZZ preserves the core structure of SZZ: fixing commits are identified and then traced back to inducing commits. Although PR-SZZ improves the identification of fixing-commit links, its tracing mechanism remains broadly similar to that of other SZZ variants and mainly introduces additional filters to reduce the number of traced commits. Consequently, the same fundamental challenges remain, including large fixes, merge-related changes, and ambiguous issue labels. Second, our qualitative analysis suggests that PR-SZZ struggles primarily in cases where the difficulty is not structural or syntactic, but semantic in nature, or where fixes are distributed across multiple files. These challenges are almost certainly affecting other SZZ variants as well, since they all rely heavily on similar tracing logic based on `git diff`. Third, the error patterns we observed, such as the over-attribution of defect-inducing status to large, noisy commits and substantial variation across projects, are consistent with known limitations of SZZ-style algorithms more generally. This suggests that these issues are not unique to PR-SZZ.

At the same time, we could not directly compare the results of this study with those of other SZZ variants because of our sampling strategy. We deliberately sampled and manually validated cases that were particularly informative for PR-SZZ, based on the datasets it produced and the model predictions learned from them. As a result, the manually labeled commits used in this study do not necessarily represent equally informative or challenging cases for models trained on datasets generated by other SZZ variants. A direct comparison would therefore require repeating the same manual validation process, including the stratified sampling procedure, for each SZZ variant under consideration.

Nevertheless, because PR-SZZ represents one of the strongest SZZ variants currently available, we argue that the main conclusions of this study should generalize conceptually to other variants. Future work should test this assumption more systematically by comparing multiple SZZ variants against a broader manually validated ground truth, with the goal of mapping their respective error profiles and understanding their implications for defect prediction research.

6.3 Implications for JIT defect prediction Research

The manual examples highlight the limitations of commit-level ground truth based on bug-fixing commits. Latent defects that are never fixed, or that are fixed under non-bug labels, remain invisible. In addition, semantics of "defect-inducing" are often fuzzy in multi-commit features and merges. This suggests that the research community

should invest more in alternative ground-truth sources (e.g. curated benchmarks, fault injection, or controlled experiments) and in methods that can tolerate ambiguous or partial labels.

The observed average inflation of precision and F1 under PR-SZZ labeling suggests that absolute performance estimates reported in SZZ-based studies should be treated with caution. When label noise is systematically biased and correlated with model features, models may appear substantially better than they truly are when judged against manual ground truth. Comparative studies that report small improvements between feature sets or algorithms may be operating within the noise induced by SZZ-style labeling.

The strong project dependency of both PR-SZZ misclassifications and model performance suggests that cross-project generalization results are sensitive to the particular mix of systems. Projects with noisy issue labels and large mixed commits may inflate false positives and obscure real improvements, whereas more disciplined projects with fine-grained bug issues and targeted fixes may produce cleaner signals. Future work should therefore report project-level results and explicitly analyses the interaction between project practices, label quality, and model performance.

6.4 Implications for the Applicability of JIT defect prediction

From a practical standpoint, these results imply that, on unseen commits, the model can prioritize risky commits in several projects, but its low precision and non-trivial miss rate limit its suitability for unseen real-world projects. Developers using such a model should expect many false alarms and should not assume that all defective commits will be caught. The performance of models highly depends on the ground truth and, therefore, the applied software development process. In high-stakes settings, where missing any defect is unacceptable, such a model should not be used as a gatekeeper (e.g., automatic rejection of commits) without additional safeguards. Instead, the model might be better suited as a triage tool helping reviewers to prioritize reviews.

Per-project analyses show that performance varies considerably. In some projects, the model attains relatively high recall with modest precision; in others, both precision and recall are lower. These differences are influenced by project size, change patterns (e.g., frequent large mixed-purpose commits), and issue labeling practices. Project-specific calibration, threshold tuning, and continuous monitoring of precision and recall are essential. If practitioners want to test defect prediction methods based on SZZ variants, they should also adopt development practices that improve the quality of SZZ-derived datasets, thereby allowing models to learn more faithful defect patterns and making evaluation studies more credible. Many of these measures do align well with industry standards and well-known git workflows. Concretely, the following guidelines help SZZ produce more accurate bug–inducing links:

- **Use fine-grained, single-purpose commits.** Avoid large mixed-purpose commits that combine feature development, refactoring, formatting, and bug fixes. Instead, separate bug fixes from feature work and refactorings into distinct commits. This reduces the number of unrelated lines touched in bug-fixing commits and makes SZZ's blame-based inference more precise.

- **Avoid noise in bug-fixing commits.** When possible, restrict bug-fix commits to the minimal set of files and lines required to resolve the defect, and move unrelated clean-ups (formatting, renaming) into separate commits. This reduces the risk that SZZ over-attributes defect-inducing status to unrelated changes.
- **Link bug-fix commits reliably to issue tracker entries.** Ensure that commits fixing defects reference the corresponding bug issue (e.g. via a standard pattern such as `BUG-1234` in the commit message). Conversely, avoid using bug labels for feature requests or enhancements. Consistent, meaningful bug labels greatly improve the quality of SZZ's ground truth.
- **Distinguish bug fixes from non-bug maintenance.** Use separate issue types or labels for refactorings, performance, documentation-only changes, and test maintenance, providing guidance on the probability of commits to be defect inducing. When non-bug issues are mislabeled as "bug", SZZ will treat their commits as defect-inducing/fixing events, polluting the defect dataset.
- **Keep merge commits and squashes transparent.** Prefer merge strategies that preserve semantically meaningful change units (e.g. avoiding huge squashed commits that aggregate many unrelated changes). Where merge commits do contain bug fixes, ensure the underlying bug-fixing commits are still identifiable, so SZZ can trace inducing commits to those rather than to opaque merge aggregates.
- **Regularly inspect SZZ output on sampled cases.** Periodically review a small stratified sample of SZZ-derived bug–inducing links to identify systematic misattribution patterns (e.g. particular modules, test-only changes, or refactoring hotspots). Use these insights to refine project conventions (e.g. commit structure, labeling) and, where feasible, adjust SZZ configuration or filters.

Adhering to such guidelines does not eliminate label noise, but it can substantially support SZZ mechanisms and improve the correctness of defect datasets. In turn, this increases the likelihood that models trained on these datasets learn meaningful defect patterns rather than project-specific artifacts of the labeling process.

7 Threats to Validity

Internal validity. The main internal threat is that manual labels might not be perfect. Although expert labeling, supported by LLM assistance and consensus meetings, produced a high-quality ground truth, some commits are inherently ambiguous and relying on commit messages/diffs without the full context could lead to misunderstandings. ChatGPT was used only as an auxiliary reviewer; prompts were fixed and outputs logged, and human adjudication was always required. Inter-rater agreement was substantial but not perfect, and our manual definition of "defect-inducing" is stricter than PR-SZZ, which may bias comparisons. We mitigated this with consensus/adjudication for disagreements, and recorded labeling rationales.

External validity. We study six large open-source projects over a six-month window. While projects vary in size and domain, results may not generalize to other projects, industrial settings, later time windows, or different workflows (e.g., trunk-based development, heavy review automation) and defect reporting practices. In fact

results of our study speak to the fact that the generalizability of research might be limited.

Construct validity. We operationalize "defect-inducing commit" using PR-SZZ and manual judgment (i.e., changes later modified by bug-fixing commits referencing defects). This may not capture the broader concept of defect-inducing change: latent defects, performance regressions, issues fixed outside the tracker, or faults fixed after the observation window may be missed, and PR-SZZ can mislabel refactorings/renames. Performance is measured at the commit level; other units (files/modules/lines) could yield different results. Comparing outcomes under PR-SZZ labels (automatic evaluation) and manual labels (sample evaluation) partially assesses this bias, but the construct remains imperfect. It is unclear that the results presented in this study generalize to other projects or SZZ variants. However, the general trend and issues discussed here are general to SZZ and should be valid in other contexts.

Conclusion validity. Manual validation covers 420 commits using stratified sampling. Unbiased population estimates depend on correct inverse-probability weighting; errors in weights or shifts in stratum distributions could bias results. We mitigate this by documenting stratum sizes and sampling fractions and using bootstrap confidence intervals, but some project-stratum cells are small, increasing variance. Small effects may not be statistically detectable within the labeled commits. The presented error class taxonomy is derived from the manually inspected commits in our validation set and may therefore not cover all failure modes that occur in other projects, time periods, or SZZ variants. Different datasets and development practices may exhibit additional error classes or different distributions of the categories identified here. Finally, we evaluate a single model configuration; alternative choices could change absolute performance, though trends and qualitative patterns with respect to PR-SZZ noise are likely to persist.

8 Conclusion and Future Work

This study provides an empirically grounded assessment of how SZZ-induced label noise affects the evaluation of JIT-DP models. By manually validating a stratified sample of commits and comparing both PR-SZZ labels and model predictions against these reference labels, we show that residual labeling errors are not merely random noise. Instead, they are directional, project-dependent, and concentrated among commits labeled as defect-inducing. As a consequence, standard SZZ-based evaluation pipelines can systematically overestimate the apparent effectiveness of JIT-DP models and inflates commonly reported metrics such as precision and F1. The effect is particularly concerning because models may also learn such labeling artifacts during training, especially when their features correlate with SZZ-internal mechanisms.

These findings have implications for both research and practice. For researchers, they show that progress in JIT-DP cannot be assessed reliably without considering label quality and uncertainty. Apparent gains from feature engineering or model complexity may be misleading if they are evaluated only against SZZ-derived labels. For practitioners, the results suggest that JIT-DP models should be used cautiously: they can provide useful signals for prioritizing review and testing efforts, but should not

serve as standalone quality gates. In practical settings, predictions should be combined with complementary analyses, project-specific calibration, and human judgment.

Future work should treat ground-truth construction as a central methodological challenge in JIT-DP research. Promising directions include targeted expert validation, developer-in-the-loop verification, and sampling-based evaluation protocols that explicitly account for residual label uncertainty. Such approaches are needed to address project-specific development practices, delayed defect discovery, latent defects, and changes whose defect status remains inherently ambiguous. Future studies should also develop standardized evaluation protocols that reflect realistic deployment conditions, including temporality, evolving project baselines, delayed verification, and limited developer attention.

Overall, reliable progress in JIT-DP depends not only on better predictive models, but also on more credible labels and more transparent evaluation practices. Without such improvements, reported advances risk reflecting better adaptation to noisy labeling pipelines rather than genuine improvements in practical defect detection.

Statements and Declarations

No funds, grants, or other support was received for conducting this study. The authors have no competing interests to declare that are relevant to the content of this article.

Data Availability

The data supporting the findings of this study is openly available in figshare at https://figshare.com/s/406d8344fd81635ad0e6.

References

[1] Peter Bludau and Alexander Pretschner. Feature sets in just-in-time defect prediction: An empirical evaluation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2022, page 22–31, New York, NY, USA, 2022. Association for Computing Machinery.

[2] Peter Bludau and Alexander Pretschner. Pr-szz: How pull requests can support the tracing of defects in software repositories. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1–12, 2022.

[3] W.G. Cochran. *Sampling Techniques*. Wiley, 1977.

[4] Daniel Alencar da Costa, Shane McIntosh, Weiyi Shang, Uirá Kulesza, Roberta Coelho, and Ahmed E. Hassan. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 43(7):641–657, 2017.

[5] Steven Davies, Marc Roper, and Murray Wood. Comparing text-based and dependence-based approaches for determining the origins of bugs. *Journal of Software: Evolution and Process*, 26(1):107–139, 2014.

[6] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, 1 edition, 1994.

[7] Siqi Fan, Xin Liu, Yingli Zhang, Yuan Tan, Luxing Yin, Zhaorun Chen, Song Li, Lei Qiao, and Rui Zhou. Llm-szz: Novel vulnerability-inducing commit identification driven by large language model and cve description. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 48–60, 2025.

[8] Yuanrui Fan, Xin Xia, Daniel Alencar da Costa, David Lo, Ahmed E. Hassan, and Shanping Li. The impact of mislabeled changes by szz on just-in-time defect prediction. *IEEE Transactions on Software Engineering*, 47(8):1559–1586, 2021.

[9] Steffen Herbold, Alexander Trautsch, Fabian Trautsch, and Benjamin Ledel. Problems with szz and features: An empirical study of the state of practice of defect prediction data collection. *Empirical Software Engineering*, 27(2):42, 2022.

[10] Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 392–401, 2013.

[11] Abram Hindle, Daniel M. German, and Ric Holt. What do large commits tell us? a taxonomical study of large commits. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, page 99–108, New York, NY, USA, 2008. Association for Computing Machinery.

[12] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952.

[13] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, 39(6):757–773, June 2013.

[14] Sunghun Kim, Thomas Zimmermann, Kai Pan, and E. James Jr. Whitehead. Automatic identification of bug-introducing changes. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pages 81–90, 2006.

[15] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. Investigating code review quality: Do people and participation matter? In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 9 2015.

[16] Sharon Lohr. *Sampling: Design and Analysis*. Chapman & Hall, 04 2019.

[17] Shane McIntosh and Yasutaka Kamei. Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 560, New York, NY, USA, 2018. Association for Computing Machinery.

[18] Audris Mockus and David M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, August 2002.

[19] Edmilson Campos Neto, Daniel Alencar da Costa, and Uirá Kulesza. Revisiting and improving szz implementations. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2019.

[20] Edmilson Campos Neto, Daniel Alencar da Costa, and Uirá Kulesza. The impact of refactoring changes on the szz algorithm: An empirical study. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 380–390, 2018.

[21] Feifei Niu, Junqian Shao, Christoph Mayr-Dorn, Liguo Huang, K.G. Wesley Assunção, Chuanyi Li, Jidong Ge, and Alexander Egyed. Refactoring != bug-inducing: Improving defect prediction with code change tactics analysis. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*, pages 527–538, 2025.

[22] Salomé Perez-Rosero, Robert Dyer, Samuel W. Flint, Shane McIntosh, and Witawas Srisa-an. Wia-szz: Work item aware szz: Wia-szz: Work item aware szz. *Empirical Softw. Engg.*, 30(3), March 2025.

[23] Christoffer Rosen, Ben Grawi, and Emad Shihab. Commit guru: analytics and risk prediction of software commits. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 966–969, New York, NY, USA, 2015. Association for Computing Machinery.

[24] Yu Shi, Hao Li, Bram Adams, and Ahmed E. Hassan. Beyond blame: Rethinking szz with knowledge graph search, 2026.

[25] Emad Shihab, Ahmed E. Hassan, Bram Adams, and Zhen Ming Jiang. An industrial study on the risk of software changes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*. ACM Press, 2012.

[26] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, page 1–5, New York, NY, USA, 2005. Association for Computing Machinery.

[27] Lingxiao Tang, Jiakun Liu, Zhongxin Liu, Xiaohu Yang, and Lingfeng Bao. Llm4szz: Enhancing szz algorithm with context-enhanced assessment on large language models. *Proc. ACM Softw. Eng.*, 2(ISSTA), June 2025.

[28] Ronaldo C. Veras, George G. Cabral, and Adriano L. I. Oliveira. Semantic szz: Mitigating the impact of misclassified corrective changes in just-in-time software defect prediction. In *2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 769–774, 2025.

[29] Chadd Williams and Jaime Spacco. Szz revisited: Verifying when changes induce fixes. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, DEFECTS '08, page 32–36, New York, NY, USA, 2008. Association for Computing Machinery.