The Ground Truth Effect: Investigating SZZ Variants in Just-in-Time Vulnerability Prediction

Alfonso Cannavale[1 [0009−0002−0209−5974]], Emanuele Iannone[2 [0000−0001−7489−9969]], Gianluca Di Lillo[1 [0000−0000−0000−0000]], Fabio Palomba[1 [0000−0001−9337−5116]], and Andrea De Lucia[1 [0000−0002−4238−1425]]

[1] Software Engineering (SeSa) Lab, University of Salerno, Italy
{acannavale, fpalomba, adelucia}@unisa.it, g.dilillo1@studenti.unisa.it
[2] Institute of Software Security, Hamburg University of Technology, Germany
emanuele.iannone@tuhh.de

Abstract. Just-in-Time (JIT) vulnerability prediction is critical for proactively securing software, yet its effectiveness heavily relies on the quality of the ground truth used for training models. This ground truth is commonly established using variants of the SZZ algorithm to identify vulnerability-contributing commits (VCCs). However, the impact of choosing a specific SZZ variant on model performance remains largely unexplored. In this study, we systematically investigate the effect of eight SZZ variants on JIT vulnerability prediction across seven open-source Java projects. Our findings reveal that the choice of the SZZ variant is a non-trivial factor. Models trained with datasets labeled by variants like **B-SZZ, V-SZZ, and VCC-SZZ** achieve strong and stable predictive performance, with **median MCC scores often exceeding 0.50**. In contrast, variants such as **L-SZZ and R-SZZ** produce models that perform no better than random chance, with **median MCC scores close to 0.0**. This performance gap demonstrates that an inappropriate SZZ variant can invalidate prediction models, underscoring the necessity of a principled approach to defining *ground truth*.

1 Introduction

Just-in-Time (JIT) vulnerability prediction aims to identify security-introducing commits before they are integrated into a codebase, providing a first line of defense in modern software development [22,19]. The performance of the machine learning models at the core of this task depends heavily on the *ground truth*—the labeled dataset of vulnerable and clean commits used for their training. Given the infeasibility of manual labeling, researchers universally rely on automated heuristics. The most established one is the SZZ algorithm [28], which identifies *vulnerability-contributing commits* (VCCs) retroactively.

However, SZZ is not a single algorithm but a family of variants, each employing different heuristics. For instance, some variants, like **B-SZZ**, rely on basic line-level history tracing, while more advanced ones, like **V-SZZ**, implement complex logic to trace a vulnerability's origin across multiple commits. This

proliferation of approaches, combined with a lack of comparative studies in the vulnerability context, creates a critical uncertainty: researchers and practitioners are left to choose a variant without understanding the downstream consequences of their choice. This raises the fundamental question of *how, and to what extent, the selection of an SZZ variant influences the resulting JIT vulnerability prediction models.*

To address this gap, we conduct a large-scale empirical study on seven Java projects, investigating the impact of eight distinct SZZ variants. We assess both the agreement between the ground truths they generate ($\mathbf{RQ}_1$) and, more importantly, the downstream effect on the performance of JIT prediction models ($\mathbf{RQ}_2$). Our results reveal a performance gap that depends on the chosen variant. We found that models built using ground truths from **B-SZZ, V-SZZ, and VCC-SZZ** are consistently effective, achieving high predictive power (median $MCC > 0.50$). Conversely, models relying on **L-SZZ and R-SZZ** completely fail, delivering performance equivalent to a random guess (median MCC ~ 0.0).

This paper provides the following contributions:

- We systematically evaluate, for the first time, the impact of eight SZZ variants on *vulnerability* prediction, demonstrating that the choice of the labeling heuristic is a critical factor.
- We provide clear, empirical evidence that certain variants (L-SZZ, R-SZZ) are unsuitable for this task and can lead to unreliable, near-random models.
- We offer actionable guidance for researchers and practitioners, identifying a set of reliable SZZ variants (B-SZZ, V-SZZ, VCC-SZZ, MA-SZZ) that provide a solid foundation for building effective JIT prediction models.

2 Background and Related Work

Research in vulnerability prediction has evolved from file-level analysis [27,32] to finer-grained, commit-level Just-in-Time (JIT) approaches. These JIT models, studied in works like those by Lomio et al. [19] and Nguyen et al. [22], offer immediate feedback but their effectiveness depends on reliable ground truth of *vulnerability-contributing commits* (VCCs). Since manual labeling is impractical, the de facto standard is to use the SZZ algorithm to automatically identify VCCs [28]. Our study builds on the dataset established by Lomio et al. [19] to investigate a critical, often-overlooked aspect: the impact of the SZZ variant choice on the final model.

The SZZ algorithm exists in numerous variants, each with distinct heuristics. The implementation and correctness of these variants have been systematically studied by Rosa et al. [26], who provided both a unified tool, PySZZ, and a developer-informed oracle to evaluate them. Their work highlights the concrete implementation differences between variants like **B-SZZ** [28], which uses simple line-based annotation, and more advanced ones like **V-SZZ** [2], tailored for vulnerabilities. Despite their widespread use, the relative impact of these variants on JIT *vulnerability* prediction remains unaddressed. A full description of

the eight variants we investigate, implemented through PySZZ, is available in Table 1.

The most related study to ours is by Fan et al. [11], who investigated the impact of SZZ variants on JIT *defect* prediction. They found that B-SZZ and MA-SZZ had minimal negative impact, while AG-SZZ degraded performance. Our work is different and complementary: (1) we focus on the more critical domain of **vulnerability prediction**, not general defects; (2) we analyze a broader and more modern set of **eight SZZ variants**, including those used in studies like Lomio et al.'s [19]; and (3) we evaluate not only model performance but also the **agreement** between the ground truths themselves. This study, therefore, provides the first comprehensive analysis of the *Ground Truth Effect* in the specific context of JIT vulnerability prediction.

3 Research Method

The *goal* of this study is to investigate how data labeling techniques impact JIT vulnerability prediction models' ground truth. The *purpose* is to understand which SZZ techniques yield better-performing and more stable JIT vulnerability prediction models. This study targets *researchers* interested in ground truth impact on JIT vulnerability prediction and *developers* who may adopt these models in their projects.

The study addresses two research questions. First, we investigate how different SZZ techniques compare in retrieving VCCs to assess their similarities, differences, and impact on ground truth construction.

Second, we assess how SZZ technique differences affect ground truth by training and validating machine learning models on resulting datasets, evaluating the impact of VCC variations on predictive performance.

> **RQ$_1$.** *How similar are the different SZZ techniques in retrieving vulnerability-contributing commits?*

> **RQ$_2$.** *How do the performances of models vary depending on the SZZ technique used to build the ground truth?*

To answer these questions, we conducted a mining software repository study, collecting VCCs using eight SZZ variants. We compared the resulting datasets to assess overlap, identify labeling discrepancies, and evaluate their impact on model performance. Figure 1 summarizes our methodology adhering to the ACM SIGSOFT Empirical Research Standards [1].

3.1 Ground Truth Construction and Experimental Setup

VCC Identification. Starting from the fixing commits identified by Lomio et al. [19], we used the **PySZZ** tool [26] to run eight distinct SZZ variants

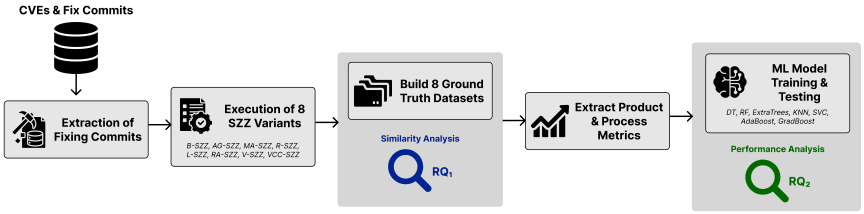


Fig. 1: Overview of our research methodology, from data collection to performance analysis.

Table 1: The eight SZZ variants analyzed in our study.

Variant	**Core Heuristic**
B-SZZ [28]	Basic 'git annotate' on deleted lines.
AG-SZZ [17]	Traverses annotation graph; ignores cosmetic changes.
MA-SZZ [8]	Like AG-SZZ, but also ignores meta-changes.
R-SZZ [9]	Like MA-SZZ, but only considers the most recent commit.
L-SZZ [8]	Like MA-SZZ, but only considers the largest commit.
RA-SZZ [21]	Like MA-SZZ, but ignores refactoring-related lines.
VCC-SZZ [16]	Advanced filtering; ignores test/build files, blames more lines.
V-SZZ [2]	Repeats 'git blame' until origin; vulnerability-specific filters.

and identify VCCs. The variants, summarized in Table 1, include both general-purpose (e.g., B-SZZ) and vulnerability-specific (e.g., V-SZZ) algorithms.

Controlled Experiment Design. For each of the eight variants, we created a distinct dataset. To isolate the effect of the labeling heuristic, these datasets differ *only* in their positive instances (the VCCs). The set of negative instances was sampled once (at a 1% ratio relative to project size, following [19]) and kept **identical** across all eight datasets. This controlled setup ensures that any observed performance difference is directly attributable to the SZZ variant used.

Model Training and Validation. We adopted two groups of commit-level metrics: *product metrics* capturing structural aspects of modified code (main Chidamber & Kemerer metrics [7]: LOC, WMC, CBO, RFC, DIT, NOC) and *process metrics* capturing code evolution and modification history (added and removed lines, modified files, author's prior commits, modification entropy, author's workload). These metrics, shown to be effective in prior vulnerability research [19,27,32,24], model developer context as factors like inexperience or high workload correlate with defect introduction [32]. Complete metric descriptions are in our online appendix [5].

The *target variable* is a binary label indicating whether each commit is a VCC (1) or not (0). This labeling was performed separately for each of the eight SZZ variants, resulting in eight ground truth versions with varying VCC counts per project.

Following prior work [19,20], we selected six machine learning algorithms: Decision Tree [4], Random Forest [3], Extra Trees [15], K-Nearest Neighbors [31], Linear Support Vector Classifier [29], AdaBoost [13], and Gradient Boosting [14]. Model training involved three stages: (1) removing collinear features using Variance Inflation Factor (VIF) to reduce overfitting risk [23]; (2) addressing class imbalance through SMOTE [6] on training data; and (3) hyperparameter optimization via Random Search [23,18]. We employed Leave-One-Group-Out (LOGO) cross-validation for cross-project evaluation, where each project serves as a test set once across seven iterations. This approach better reflects operational conditions where models must generalize to unseen projects.

3.2 Data Analysis Protocol

To address $\mathbf{RQ}_1$, we analyzed SZZ variant overlap using *Jaccard similarity* [25] between VCC sets for each vulnerability-fix pair, creating distributions rather than single scores. We applied Wilcoxon signed-rank tests [30] comparing these distributions against baselines of total disagreement (all zeros) and perfect agreement (all ones). A $p < 0.05$ indicates a significant difference from the baseline.

To address $\mathbf{RQ}_2$, we evaluated model performance using the **Matthews Correlation Coefficient (MCC)**, which provides a balanced measure suitable for imbalanced datasets. We applied *Friedman tests* [10] for each ML algorithm to determine if SZZ variant choice significantly impacts performance, followed by post-hoc *Nemenyi tests* [10] when significant differences were found ($p < 0.05$).

4 Results and Discussion

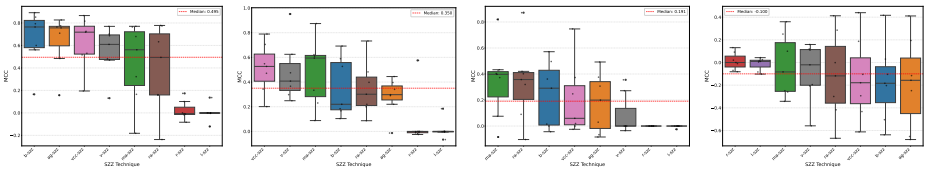
4.1 $\mathbf{RQ}_1$: How similar are the SZZ variants?

To answer RQ_1, we measured the overlap between the VCC sets generated by each pair of SZZ variants using the Jaccard similarity. Our statistical analysis confirms that while the variants are not completely disjoint, they are also not interchangeable ($p < 0.05$ against both baselines of 0 and 1, see online appendix [5]).

The descriptive analysis reveals the practical extent of this divergence. On average, any two variants agree on only half of the VCCs they identify (mean Jaccard index ˜0.50, std. dev. 0.30). The agreement ranges from a high of 0.89 for the most similar pair to a low of 0.29 for the most dissimilar.

Key Findings for RQ_1

SZZ variants are only moderately similar and cannot be used interchangeably. On average, two variants agree on only 50% of the identified vulnerable commits, creating substantially different ground truths for training prediction models.



(a) Gradient Boosting (Sensitive) (b) Decision Tree (Sensitive) (c) Random Forest (Robust) (d) Linear SVC (Insensitive)

Fig. 2: MCC distributions for four representative classifiers. Variants like L-SZZ and R-SZZ consistently yield poor models (MCC $\sim$ 0.0), while B-SZZ, V-SZZ, etc., lead to effective ones (MCC $>$ 0.5).

4.2 RQ_2: What is the impact on model performance?

To answer RQ_2, we trained seven machine learning models on the eight distinct ground truths and evaluated their performance using the Matthews Correlation Coefficient (MCC). Figure 2 shows the MCC distributions for four representative classifiers. The results reveal a performance gap driven by the SZZ variant. A clear pattern emerges across all models: ground truths from **L-SZZ** and **R-SZZ** consistently lead to models with near-random performance, with median MCC scores close to 0.0. In contrast, variants like **B-SZZ, V-SZZ, MA-SZZ, and VCC-SZZ** consistently produce effective models, with median MCC scores often exceeding 0.50, especially for sensitive classifiers like Gradient Boosting and Decision Tree.

To validate these observations, we performed a Friedman test for each classifier. As shown in Table 2, the choice of SZZ variant has a statistically significant impact ($p < 0.05$) on the performance of the most sensitive models (e.g., Decision Tree, Gradient Boosting, Random Forest). Post-hoc Nemenyi tests (detailed in the online appendix [5]) confirm that for these models, the performance degradation caused by L-SZZ and R-SZZ is statistically significant when compared to top-performing variants like B-SZZ and VCC-SZZ.

Key Findings for RQ_2

The choice of the SZZ variant is a factor that can make the difference between an effective prediction model and a useless one. Inappropriate variants (L-SZZ, R-SZZ) lead to models with near-random performance (MCC ˜0.0), while other variants (B-SZZ, V-SZZ, VCC-SZZ, MA-SZZ) enable models to achieve high predictive power (median MCC $>$ 0.50).

4.3 Discussion and Implications

Our findings have direct implications for researchers and practitioners. The most critical one is a clear warning: **using L-SZZ or R-SZZ for JIT vulnerability prediction is not efficient**. These variants, relying on overly simplistic

Table 2: Friedman test results ($p < 0.05$) on MCC scores. A ✓ indicates a significant performance difference across SZZ variants.

Classifier	MCC Significant?
AdaBoost	NS
Decision Tree	✓
Extra Trees	NS
Gradient Boosting	✓
K-Nearest Neighbors	NS
Linear SVC	NS
Random Forest	✓

heuristics, systematically produce noisy ground truths that prevent models from learning meaningful patterns.

Conversely, our results provide a set of **recommended variants: B-SZZ, V-SZZ, MA-SZZ, and VCC-SZZ**. These provide a more reliable basis for model training. Notably, the original B-SZZ remains a strong baseline, suggesting that complexity is not always a synonym for better performance in this context.

Furthermore, the impact is **mediated by the ML model**. Sensitive classifiers like Gradient Boosting amplify the differences in ground truth quality, while robust or simpler ones like LinearSVC are less affected (though often at the cost of lower overall performance). This implies that the reported performance of a new JIT prediction technique is heavily biased by the chosen SZZ variant. Comparing studies that use different SZZ variants without acknowledging this effect can lead to flawed conclusions.

5 Threats to Validity

VCC retrieval accuracy represents a key threat, stemming from inherent SZZ limitations rather than implementation issues. We mitigated this by selecting diverse SZZ variants, including both general-purpose (B-SZZ) and vulnerability-specific techniques (V-SZZ). This study evaluates SZZ's impact on model performance without validating absolute VCC correctness against manual ground truth; effectiveness is judged solely by downstream ML model performance.

Our ML pipeline introduces potential confounding factors. The 1% negative sampling ratio, while based on prior work [19], and SMOTE balancing, despite known weaknesses [12], could affect results. We mitigated these threats by applying consistent preprocessing across all experiments.

Our focus on open-source Java projects limits the external validity of our findings. While Java was chosen as the most studied language in SZZ research, results may not generalize to other languages or industrial settings with different development dynamics.

6 Conclusion and Future Work

Our findings reveal that differences in SZZ variant VCC identification lead to statistically significant variations in model performance. While variants share core VCCs ($\mathbf{RQ}_1$), ground truth divergences affect just-in-time vulnerability prediction ($\mathbf{RQ}_2$), particularly in data-sensitive models. B-SZZ, V-SZZ, MA-SZZ, and VCC-SZZ produce more stable models with median MCC scores often exceeding 0.50, while L-SZZ and R-SZZ yield weaker outcomes near 0.0. The impact varies by algorithm: decision trees and boosting models amplify labeling differences, while KNN shows less sensitivity but worse overall performance.

Practitioners should interpret these results cautiously, as model reliability remains tied to the unverified accuracy of SZZ ground truth labeling. Future work should investigate generalization beyond Java projects, integrate deep learning models with richer feature sets, and explore hybrid VCC retrieval techniques beyond SZZ heuristics.

Acknowledgment

This work was partially supported by projects SERICS (PE00000014) and FAIR (PE0000013) under the NRRP MUR program funded by the EU - NGEU, and the EU-funded project Sec4AI4Sec (grant no. 101120393).

References

1. Ralph et al., P.: Empirical standards for software engineering research (2021)
2. Bao, L., Xia, X., Hassan, A.E., Yang, X.: V-szz: automatic identification of version ranges affected by cve vulnerabilities. In: Proceedings of the 44th international conference on software engineering. pp. 2352–2364 (2022)
3. Breiman, L.: Random forests. Machine learning **45**, 5–32 (2001)
4. Breiman, L., Friedman, J., Olshen, R.A., Stone, C.J.: Classification and regression trees. Routledge (2017)
5. Cannavale, A., Iannone, E., Di Lillo, G., Palomba, F., De Lucia, A.: The ground truth effect: Dataset and materials (supplementary) (2025). https://doi.org/10.6084/m9.figshare.28788857, https://doi.org/10.6084/m9.figshare.28788857
6. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research **16**, 321–357 (2002)
7. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on software engineering **20**(6), 476–493 (1994)
8. Da Costa, D.A., McIntosh, S., Shang, W., Kulesza, U., Coelho, R., Hassan, A.E.: A framework for evaluating the results of the szz approach for identifying bug-introducing changes. IEEE Transactions on Software Engineering **43**(7), 641–657 (2016)
9. Davies, S., Roper, M., Wood, M.: Comparing text-based and dependence-based approaches for determining the origins of bugs. Journal of Software: Evolution and Process **26**(1), 107–139 (2014)

10. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research **7**(Jan), 1–30 (2006)
11. Fan, Y., Xia, X., Da Costa, D.A., Lo, D., Hassan, A.E., Li, S.: The impact of mislabeled changes by szz on just-in-time defect prediction. IEEE transactions on software engineering **47**(8), 1559–1586 (2019)
12. Fernández, A., Garcia, S., Herrera, F., Chawla, N.V.: Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. Journal of artificial intelligence research **61**, 863–905 (2018)
13. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences **55**(1), 119–139 (1997)
14. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
15. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine learning **63**, 3–42 (2006)
16. Iannone, E., Guadagni, R., Ferrucci, F., De Lucia, A., Palomba, F.: The secret life of software vulnerabilities: A large-scale empirical study. IEEE Transactions on Software Engineering **49**(1), 44–63 (2022)
17. Kim, S., Zimmermann, T., Pan, K., James Jr, E., et al.: Automatic identification of bug-introducing changes. In: 21st IEEE/ACM international conference on automated software engineering (ASE'06). pp. 81–90. IEEE (2006)
18. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research **18**(185), 1–52 (2018)
19. Lomio, F., Iannone, E., De Lucia, A., Palomba, F., Lenarduzzi, V.: Just-in-time software vulnerability detection: Are we there yet? Journal of Systems and Software **188**, 111283 (2022)
20. Morrison, P., Herzig, K., Murphy, B., Williams, L.: Challenges with applying vulnerability prediction models. In: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. pp. 1–9 (2015)
21. Neto, E.C., Da Costa, D.A., Kulesza, U.: 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). pp. 380–390. IEEE (2018)
22. Nguyen, S., Nguyen, T.T., Vu, T.T., Do, T.D., Ngo, K.T., Vo, H.D.: Code-centric learning-based just-in-time vulnerability detection. Journal of Systems and Software **214**, 112014 (2024)
23. O'brien, R.M.: A caution regarding rules of thumb for variance inflation factors. Quality & quantity **41**, 673–690 (2007)
24. Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., Acar, Y.: Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 426–437 (2015)
25. Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press (2011)
26. Rosa, G., Pascarella, L., Scalabrino, S., Tufano, R., Bavota, G., Lanza, M., Oliveto, R.: Evaluating szz implementations through a developer-informed oracle. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). pp. 436–447. IEEE (2021)

27. Shin, Y., Williams, L.: An empirical model to predict security vulnerabilities using code complexity metrics. In: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. pp. 315–317 (2008)
28. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? ACM sigsoft software engineering notes **30**(4), 1–5 (2005)
29. Vapnik, V.: Support-vector networks. Machine learning **20**, 273–297 (1995)
30. Woolson, R.F.: Wilcoxon signed-rank test. Encyclopedia of biostatistics **8** (2005)
31. Zhang, Z.: Introduction to machine learning: k-nearest neighbors. Annals of translational medicine **4**(11) (2016)
32. Zimmermann, T., Nagappan, N., Williams, L.: Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In: 2010 Third International Conference on Software Testing, Verification and Validation. pp. 421–428 (2010). https://doi.org/10.1109/ICST.2010.32