

Original Research

Received 28 June 2026

Revised 02 Aug 2026

Accepted 15 Aug 2026

Natural Resources for Human Health 2026, Vol. 6 (17s): 684–699

KEYWORDS:

Just-In-Time Software Defect Prediction; Temporal Robustness; Distribution Shift; Random Forest; Online JIT-SDP; Chronological Evaluation

Temporal Robustness and Lightweight Adaptation for Just-In-Time Software Defect Prediction

¹D. Pavan Hari Krishna, ²Sonti Thirupathamma, ³Dr. M. V. D. Prasad

¹Student, M.Tech, Department of CSE KL University, Vaddeswaram, India
daggumatipavanharikrishna@gmail.com

²Ph.D. Research Scholar, Department of ECE SRM University–AP, Amaravati, India
thirupathamma_sonti@srmap.edu.in
ORCID: 0009-0009-2768-8064

³Associate Professor, Department of ECE KL University, Vaddeswaram, India
Corresponding Author mvd_ece@kluniversity.in

ABSTRACT: Just-In-Time Software Defect Prediction (JIT-SDP) is commonly evaluated using historical software changes, yet deployment conditions are inherently temporal. This study investigates temporal robustness and lightweight within-project adaptation using a leakage-resistant chronological protocol. The original Brackets case study contains 11,601 changes and 14 change-level metrics; a strict 70/30 split yields 8,120 historical and 3,481 future observations. Recursive Feature Elimination (RFE), fitted only within historical development data, selects six features. The same protocol is then extended to four additional processed JIT-SDP streams from Tomcat, JGroups, Spring-Integration, and Camel. On Brackets, the frozen Random Forest obtains F1=0.5366, ROC-AUC=0.7961, and PR-AUC=0.5161. On the four additional projects, frozen F1 ranges from 0.2975 to 0.6741, demonstrating substantial project-level variation. Temporal segmentation shows marked future-period degradation in three of the four additional projects. Cumulative and fixed recency windows, together with a 4,000-record recency model using adaptive threshold selection, are compared without using future-test labels for model selection. Across the 16 project-period blocks, the Friedman test detects differences among strategies ($p < 0.0001$, Kendall's $W = 0.3271$), but pairwise results do not establish a uniformly superior adaptation strategy; the 2,000-record window is significantly different from the frozen strategy ($p = 0.0042$) in the tested direction of lower mean F1. A published-baseline-style comparator produces mixed project-level differences and is explicitly distinguished from an exact reproduction of BORB. Bootstrap confidence intervals further show overlapping uncertainty between frozen and adaptive-threshold configurations for each project. The evidence supports temporal degradation and project-dependent adaptation behavior, rather than a claim of universal adaptation superiority.

1 INTRODUCTION

Software defects increase maintenance cost and can affect reliability, security, and operational stability. Just-In-Time Software Defect Prediction (JIT-SDP) predicts whether an individual software change is likely to introduce a defect and can therefore support review and testing prioritization [1,2]. A central methodological issue is that software repositories evolve. Developer participation, coding practices, project structure, change size, and defect-generating processes can change over time. Consequently, a model trained on historical changes may encounter a different statistical environment after deployment. Random train–test partitioning can obscure this issue because observations from later periods may influence model

development. Chronological evaluation is therefore important when the research question concerns future prediction [3,4]. Prior JIT-SDP research has already established that temporal ordering and online learning matter. Cabral et al. investigated online and offline learning for online JIT-SDP across multiple open-source projects [3], while Cabral and Minku studied reliability issues in online JIT-SDP and the effects of verification latency [5]. Deep learning approaches such as DeepJIT also demonstrate that richer change representations can improve JIT-SDP modelling [6], and subsequent work has investigated how far such representations generalize [7]. These studies make clear that the present work should not be interpreted as introducing a new JIT-SDP learner. Instead, this paper addresses a narrower empirical question: how much does predictive behavior change when a conventional JIT-SDP model is evaluated strictly in the future, how measurable is the associated feature-distribution shift, and do lightweight within-project retraining strategies provide consistent benefits? The study combines chronological evaluation, feature-level shift analysis, period-wise performance analysis, and controlled recency-based adaptation under a common Random Forest learner. The experimental program also extends the original Brackets case study to four additional JIT-SDP projects, thereby testing whether the temporal observations are project-specific or recur across multiple repositories.

The contributions are deliberately empirical:

- A leakage-resistant chronological evaluation protocol separating historical development from genuinely future observations.
- A multi-project replication covering Brackets, Tomcat, JGroups, Spring-Integration, and Camel using common change-level features where compatible.
- Quantification of historical-to-future feature distribution shift using Kolmogorov–Smirnov statistics and Population Stability Index for the original Brackets analysis.
- Period-wise measurement of temporal degradation across future observations rather than reporting only one aggregate test score.
- A controlled comparison of frozen, cumulative, sliding-window, and adaptive-threshold retraining strategies.
- Bootstrap uncertainty estimates and non-parametric statistical testing to distinguish practical differences from evidence of statistical superiority.
- An explicit comparison with a published-baseline-style implementation, together with a transparent statement of the implementation differences that prevent it from being described as an exact BORB reproduction.

2 RELATED WORK

2.1 JIT-SDP and Change-Level Metrics

Kamei et al. established a large-scale empirical foundation for JIT-SDP using change-level metrics such as code churn and developer-related characteristics [1]. McIntosh and Kamei subsequently showed that predictive relationships can change over the lifetime of a project, motivating longitudinal evaluation [2].

2.2 Online and Temporal JIT-SDP

Chronological and online evaluation has also been examined directly in earlier JIT-SDP studies. Jahanshahi et al. investigated chronology, recent-data weighting, and longitudinal changes across multiple projects [14]. Tabassum et al. studied cross-project learning in realistic online JIT-SDP settings [15]. Cabral et al. compared online and offline learning models in an online JIT-SDP setting across ten open-source projects [3]. Cabral and Minku further examined reliable online JIT-SDP under verification latency and changing defect-generating conditions [5]. Song et al. questioned retrospective evaluation procedures and emphasized chronology and realistic verification delays [4].

2.3 Deep and Adaptive Approaches

DeepJIT represents software changes using learned representations rather than only conventional change metrics [6]. Zeng et al. subsequently investigated the robustness of deep JIT-SDP and reported important limitations concerning generalization and evaluation [7]. Pornprasit and Tantithamthavorn's JITLine study further showed that apparently strong deep JIT-SDP results can change substantially when future data are excluded from training, reinforcing the need for deployment-faithful evaluation [16]. More recent work has explored dynamic transfer learning, non-stationary data streams, and human-in-the-loop online

prediction [8–10]. These approaches are more elaborate than the lightweight within-project adaptation considered here. Published online baselines also include confidence-aware online sampling and prediction-based drift adaptation. The ODaSC family uses online Hoeffding-tree ensembles with adaptive resampling and label-confidence handling, while PBSA was proposed specifically to react to concept drift using prediction behaviour under verification latency [5]. These methods are important reference points for reliability-oriented online JIT-SDP. They are not numerically reproduced in the present common-protocol experiments because their original implementations, stream-processing assumptions, and online learner stacks differ from the Random-Forest chronological protocol used here. Instead, they are explicitly covered in the baseline-fidelity discussion so that the scope of the experimental comparison is not overstated.

2.4 Positioning of the Present Study

The present work does not claim novelty in chronological evaluation, sliding-window learning, or online JIT-SDP. Those ideas are established in prior literature. Its contribution is instead a controlled empirical synthesis: the same classical learner and feature representation are evaluated under a future-oriented protocol, feature shift is measured explicitly in the original Brackets case study, future performance is segmented into chronological periods, and lightweight adaptation is compared across multiple projects. The multi-project results are used to assess whether observed adaptation behavior is consistent rather than to claim a new algorithm.

3 RESEARCH QUESTIONS

3.1 RQ1: Feature Selection

How does RFE-based reduction of the conventional 14-feature representation affect predictive performance under chronological evaluation?

3.2 RQ2: Distribution Shift

Does the future portion of the original Brackets case study exhibit measurable feature-distribution shift?

3.3 RQ3: Temporal Performance

How does predictive performance change across chronological future periods and across projects?

3.4 RQ4: Temporal Adaptation

Do cumulative, sliding-window, and adaptive-threshold retraining strategies provide consistent benefits over a frozen model?

Table 1: Positioning Relative to Representative JIT-SDP Studies

Study	Focus	Learning setting	Relationship to this study
Kamei et al. [1]	Change-level JIT-SDP	Conventional supervised learning	Provides the metric foundation used by this study.
McIntosh and Kamei [2]	Longitudinal JIT-SDP	Longitudinal evaluation	Motivates temporal performance analysis.
Cabral et al. [3]	Online vs. offline JIT-SDP	Online learning	Establishes the relevance of training recency and chronology.
DeepJIT [6]	Learned change representations	Deep learning	Represents a model family not reproduced in the present common-protocol experiments.
Zeng et al. [7]	Reliability of deep JIT-SDP	Deep learning/evaluation	Highlights generalization and evaluation concerns.

Present study	Temporal robustness and lightweight adaptation	Chronological within-project evaluation	Combines shift measurement, period-wise degradation, and controlled lightweight adaptation across five projects.
---------------	--	---	--

Table 2: Multi-Project Dataset Audit

Project	Rows	Future test	Bug rate in test
Brackets	11,601	3,481	0.3402
Tomcat	233,195	5,673	0.2754
JGroups	232,684	5,498	0.1721
Spring-Integration	232,002	2,625	0.2666
Camel	234,736	9,173	0.2046

4 DATASETS AND EXPERIMENTAL PROTOCOL

4.1 Original Brackets Case Study

The Brackets dataset contains 11,601 change records and 14 conventional change-level features: fix, ns, nd, nf, entropy, la, ld, lt, ndev, age, nuc, exp, rexp, and sexp. The exact CSV used in the original analysis contains no missing values or duplicate rows. Its SHA-256 checksum is 72d3d69c337a7dd7693363e4c368c459878ef989be6d e3a0e37b8aa1081e81fc.

4.2 Additional Projects

Four additional processed JTT-SDP streams were obtained from the CPJITSDP-style data collection: Tomcat, JGroups, Spring-Integration, and Camel. They share the 14 standard change-level metrics and contain additional stream-control attributes, including project identifier and commit type. The processed streams encode delayed label availability and testing/training events; therefore, they were not treated as generic independent rows. For within-project evaluation, the target project's test events were identified using the documented commit-type semantics, while cross-project training events were excluded. The additional streams contain 32 exact duplicate rows in their raw processed files. These were removed consistently during the target-project training/test construction because the duplicates arise from the processed online-stream representation rather than from distinct additional software changes. This preprocessing decision was applied before chronological splitting and is recorded in the replication artifacts.

4.3 Chronological Evaluation

For the Brackets study, the first 8,120 chronological observations form the historical portion and the final 3,481 observations form the future test set. Within the historical portion, 6,496 records are used for development and 1,624 for validation. Feature selection and hyperparameter selection are performed only within the historical data. For the four additional projects, the target-project test events are chronologically divided using the same 70/30 future-oriented protocol after respecting the processed stream's event semantics. The resulting future test sizes are 5,673, 5,498, 2,625, and 9,173 for Tomcat, JGroups, Spring-Integration, and Camel respectively.

4.4 Temporal Segmentation

Each future test set is divided into four consecutive approximately equal-sized periods. This permits direct measurement of future-period degradation while preserving temporal order.

5 FEATURE SELECTION AND MODELS

For the Brackets case study, RFE uses a Random Forest estimator and is fitted only on the historical development data. Candidate subset sizes are evaluated on the historical validation data using F1. The final Brackets subset contains *lt*, *la*, *rexp*, *exp*, *ld*, and *nuc*. The controlled base learner is Random Forest with 100 trees, maximum depth 5, minimum leaf size 4, balanced class weighting, square-root feature selection, and random seed 42. A sensitivity analysis was added to make the RFE decision auditable. The RFE ranking was generated on the historical development fold only, and validation F1 was then measured for the ranked subsets. With the same chronological development/validation split, the sixfeature subset produced validation F1=0.6405, while the five-feature subset produced 0.6513; the corresponding six- and five-feature ROC-AUC values were 0.8136 and 0.8108. Thus, six features should not be interpreted as a

Table 3: RFE Sensitivity on the Brackets Historical Validation Fold

Features	F1	ROC-AUC	MCC
14	0.6142	0.8075	0.4460
12	0.6217	0.8119	0.4500
10	0.5946	0.8055	0.4342
8	0.6341	0.8086	0.4460
7	0.6345	0.8103	0.4477
6	0.6405	0.8136	0.4521
5	0.6513	0.8108	0.4669
4	0.6246	0.7972	0.4271
3	0.6061	0.7863	0.3993
2	0.5630	0.7434	0.3244

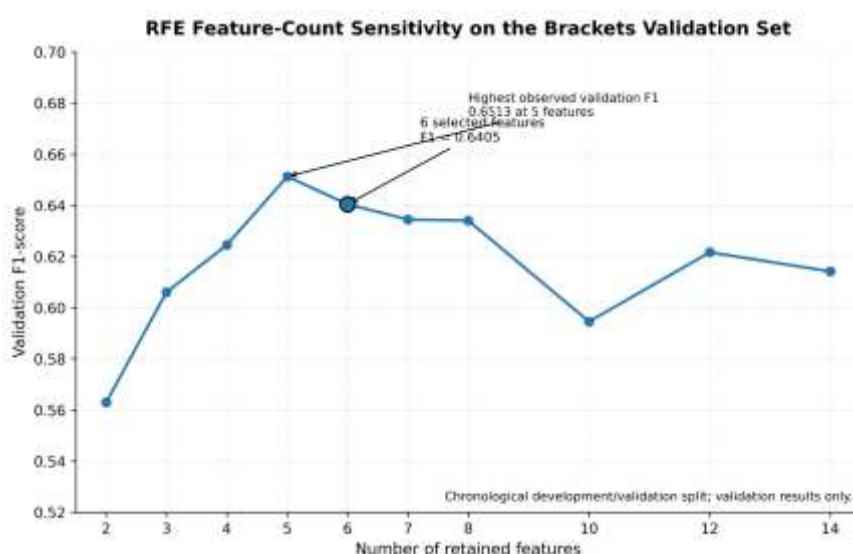


Figure 1: RFE feature-count sensitivity on the Brackets historical validation fold. The six-feature configuration used in the downstream temporal analysis is explicitly marked; the figure does not imply that six features are uniquely optimal.

uniquely optimal dimensionality. Figure 1 provides the corresponding feature-count sensitivity curve. Six was retained as the controlled representation because it is the subset selected by the original RFE ranking and preserves the study's pre-specified compact representation across the downstream temporal analyses. A permutation importance analysis on the Brackets validation fold provides an additional interpretability check: among the six selected variables, *la* has the largest mean F1 permutation importance (0.2787), followed by *lt* (0.0077), *ld* (0.0068), *nuc* (0.0055), *exp* (0.0020), and *rexp* (0.0008). These importance values are model- and validation-split-specific rather than causal effect estimates. In this study, “lightweight” refers to the use of a conventional tabular learner with a small fixed feature representation and retraining windows, rather than to a claim of universally minimal runtime. The original Brackets hyperparameter search sampled 30 configurations from 810 possible combinations using three expanding chronological validation folds. Random search was retained because the objective was controlled temporal evaluation under a fixed search budget rather than exhaustive hyperparameter optimization. XGBoost is used as a static comparator on Brackets. A separate Brackets temporal XGBoost analysis is also reported below so that the exclusion of XGBoost from the main multi-project adaptation suite does not imply that its temporal behavior is unknown.

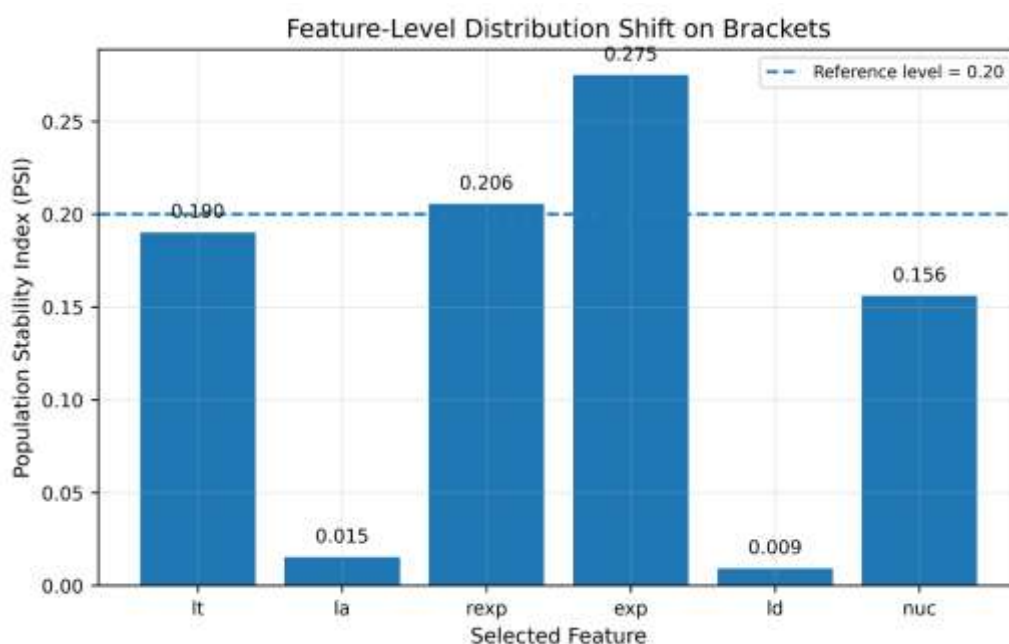


Figure 2: Population Stability Index (PSI) for the six RFE-selected Brackets features. The dashed PSI=0.20 line is shown only as a commonly used reference level for interpretation, not as a universal decision threshold.

Random Forest remains the controlled primary learner so that changes in adaptation strategy are not confounded with changes in learner family. For the four additional projects, the same learner family and evaluation philosophy are retained. Project-specific RFE is fitted only on historical development data, and the resulting feature subset is frozen before future evaluation. Cumulative retraining means that, at each temporal boundary, a new Random Forest is trained from scratch on all labeled observations available up to that boundary; no warm-start or incremental parameter update is used.

6 Distribution Shift Analysis

Distribution shift is evaluated in detail for the original Brackets case study. Historical and future distributions are compared using the two-sample Kolmogorov–Smirnov test and Population Stability Index (PSI). Benjamini–Hochberg correction is applied across the six selected feature KS tests. The six selected features show significant historical-to-future changes. Figure 2 visualizes the corresponding PSI values. The largest PSI values are observed for *exp* (0.2750), *rexp* (0.2056), and *lt* (0.1902). The corresponding KS statistics are 0.1375 for *exp*, 0.1867 for *rexp*, and 0.1561 for *lt*, with corrected *q*-values below 10^{−39} for these three variables.

7 FROZEN MULTI-PROJECT EVALUATION

Table 4 reports the frozen future-test results. Performance varies considerably across projects. SpringIntegration obtains the highest F1 (0.6741), while Camel obtains the lowest (0.2975). Figure 3 visualizes this project-level spread. This spread demonstrates why a single-project result should not be interpreted as a general estimate of JIT-SDP performance. For the original Brackets case study, the frozen Ran4

Table 4: Frozen Chronological Evaluation on the Four Additional Projects

Project	Train	Test	Accuracy	Precision	Recall	F1	ROC-AUC
Tomcat	11,793	5,673	0.7137	0.3424	0.6847	0.4565	0.7706
JGroups	12,139	5,498	0.6559	0.2027	0.7011	0.3145	0.7456
Spring-Integration	5,041	2,625	0.7653	0.5713	0.8219	0.6741	0.8618
Camel	20,427	9,173	0.7086	0.1884	0.7075	0.2975	0.7886

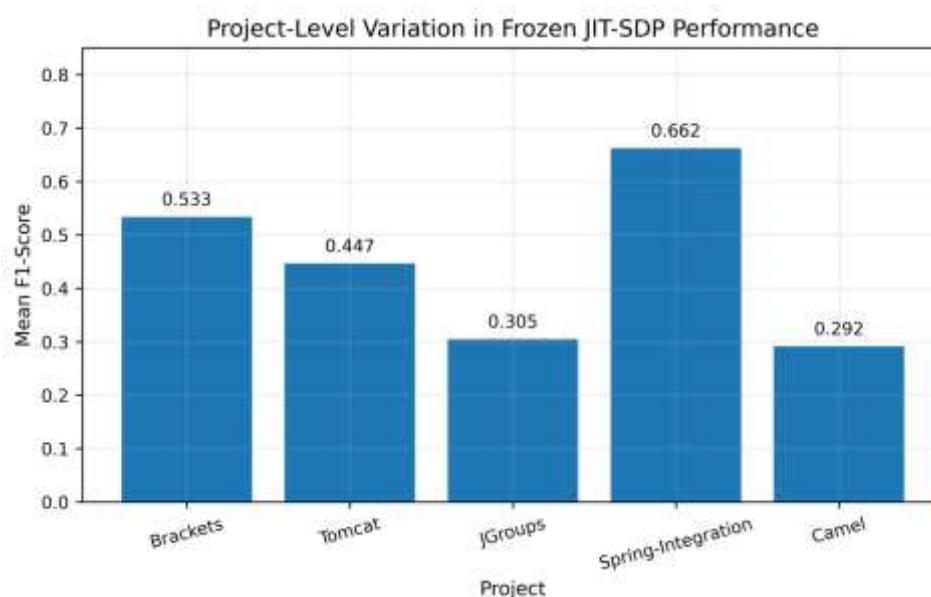


Figure 3: Frozen chronological F1-score across the four additional projects. The figure highlights substantial project-level variation under the same Random Forest evaluation protocol.

dom Forest obtains accuracy 0.7236, precision 0.4204, recall 0.7417, F1 0.5366, ROC-AUC 0.7961, PR-AUC 0.5161, and MCC 0.3900.

8 TEMPORAL PERFORMANCE ACROSS FUTURE PERIODS

Table 5 and Figure 4 summarize F1 over four future periods. Three projects show clear T1-to-T4 degradation, while JGroups shows a non-monotonic pattern. The results therefore support temporal instability, but not a universal monotonic decline. The Brackets case study shows the same phenomenon: frozen F1 falls from 0.5945 in T1 to 0.4872 in T4, a reduction of 0.1073 points. Thus, temporal degradation is observed both in the original case study and in most additional projects. Defect prevalence is reported alongside temporal performance because F1 is sensitive to the class distribution. The Brackets historical and full-future defect rates are 0.3936 and 0.2157, respectively; its T1–T4 rates are 0.2560, 0.2218, 0.2138, and 0.1713. For the additional projects, the historical/future rates are 0.3182/0.1756 for Tomcat, 0.1976/0.1126 for JGroups, 0.2544/0.2952 for Spring-Integration, and 0.2549/0.0872 for Camel. The corresponding T1–T4 future rates are shown in Table 6 and Figure 5.

9 Temporal Adaptation

Four adaptation families are evaluated: cumulative retraining, sliding windows of 2,000, 4,000, 6,000, and 8,000 recent labeled records, and a 4,000-record model with adaptive threshold selection. The adaptive configuration uses the latest 1,000 available labeled records for threshold calibration, searches thresholds from 0.20 to 0.80 in 0.05 increments using calibration F1, and then retrain on all 4,000 records. Future-test labels are never used for threshold selection. The results are project-dependent. Figures 6 and 7 provide complementary visual comparisons of these strategies. The adaptive-threshold configuration is competitive for Camel and JGroups, but it does not consistently improve over cumulative or fixed-window retraining across projects. For JGroups, for example, its mean F1 (0.3135) is below cumulative retraining (0.3294) but

Table 5: Frozen F1 Across Chronological Future Periods

Project	T1	T2	T3	T4	T1–T4 change
Camel	0.3911	0.2729	0.2972	0.2053	-0.1858
JGroups	0.2686	0.2186	0.4284	0.3038	+0.0352
Spring-Integration	0.7778	0.7587	0.7292	0.3836	-0.3941
Tomcat	0.5206	0.5046	0.3785	0.3828	-0.1379

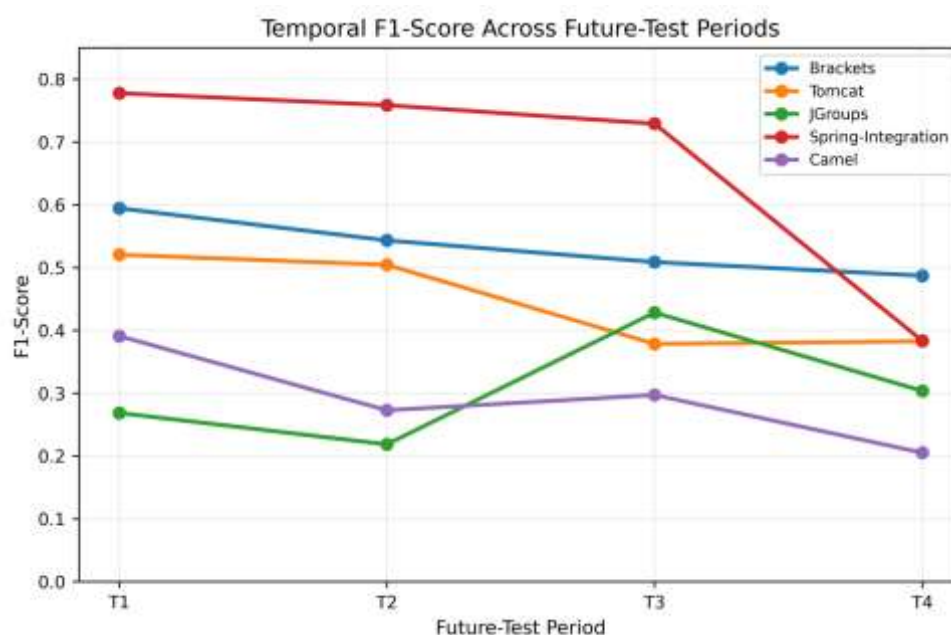


Figure 4: Frozen F1-score across four chronological future periods for Brackets, Tomcat, JGroups, Spring-Integration, and Camel. The curves show project-dependent temporal behavior rather than a universal monotonic degradation pattern.

above all fixed sliding windows. For Spring-Integration, adaptive thresholding (0.6428) is below the cumulative and longer-window configurations. No single adaptation strategy is consistently best across all projects. As a learner-specific sensitivity check, the Brackets future periods were also evaluated with XGBoost using the same six RFE-selected features and chronological training history. XGBoost F1 across T1–T4 was 0.5586, 0.5044, 0.4810, and 0.4780, with ROC-AUC values of 0.7870, 0.7586, 0.7481, and 0.7842. These results confirm that the temporal phenomenon is not unique to the Random Forest implementation; Figure 8 shows the corresponding period-wise comparison. although the multi-project adaptation suite remains Random-Forest-controlled for comparability. Class-imbalance sensitivity was also evaluated on the Brackets frozen future test. Balanced class weighting yielded F1=0.5366, ROC-AUC=0.7961, and MCC=0.3900. SMOTE yielded F1=0.5363,

ROCAUC=0.7943, and MCC=0.3896, while ADASYN yielded F1=0.5256, ROC-AUC=0.7958, and MCC=0.3764. Resampling was fitted only on the historical training data. The close SMOTE result and lower ADASYN F1 support retaining class weighting as the primary controlled treatment rather than introducing synthetic sampling into every temporal experiment. Runtime was measured on the verified local execution environment as an implementation-level sensitivity check. Across the four Brackets temporal boundaries, mean Random Forest training/prediction time per update was approximately 0.331 s for cumulative retraining, 0.144 s for a 2,000-record window, 0.176 s for 4,000, 0.246 s for 6,000, and 0.275 s for 8,000. The 4,000-record adaptive-threshold configuration averaged approximately 0.358 s per update because it performs calibration and a final retraining step. These values are hardware- and implementation-dependent and are reported to operationalize the term “lightweight”, not as a universal runtime benchmark. Threshold-calibration sensitivity was evaluated using calibration windows of 500, 1,000, and 2,000 observations while keeping the 4,000-record training window fixed. Mean F1 was 0.5329, 0.5399, and 0.5380, respectively. Thus, the selected 1,000-record calibration window was not uniquely optimal, but its performance was within the observed sensitivity range and was retained as the

Table 6: Defect Rates Across Chronological Future Periods

Project	Historical	Future	T1	T2	T3	T4
Brackets	0.3936	0.2157	0.2560	0.2218	0.2138	0.1713
Tomcat	0.3182	0.1756	0.2340	0.2193	0.1354	0.1135
JGroups	0.1976	0.1126	0.1084	0.0844	0.1820	0.0757
Spring-Integration	0.2544	0.2952	0.2100	0.3643	0.4527	0.1540
Camel	0.2549	0.0872	0.1277	0.0986	0.0754	0.0471

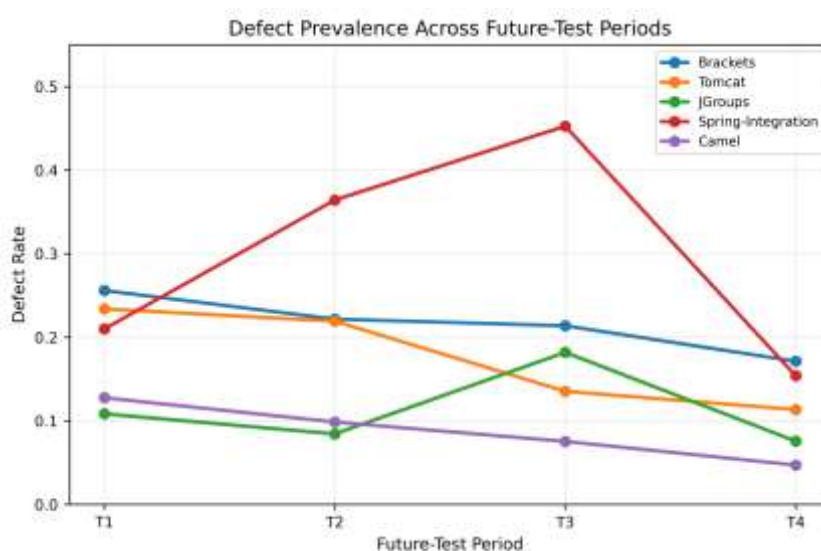


Figure 5: Defect prevalence across chronological future periods. Changes in defect prevalence provide context for interpreting period-wise F1 because F1 depends on class distribution.

primary configuration because it was specified in the original experimental protocol.

10 PUBLISHED-BASELINE-STYLE COMPARISON

To address the reviewer request for an online JIT-SDP baseline, a published-baseline-style comparator was implemented using the processed CPJITS DP-style streams and the study’s controlled Random Forest configuration. This experiment should not be described as an exact reproduction of BORB because the original BORB framework supports configurable base learners

and an online oversampling/retraining procedure that is not identical to the present implementation [3]. The comparator is therefore used as a contextual baseline rather than as proof of algorithmic superiority. The comparator has a higher mean F1 by 0.0133, but the difference is small and mixed across projects. The frozen model has a slightly higher mean ROC-AUC, whereas the comparator has a slightly higher mean MCC. These results do not support a universal ranking between the two configurations.

10.1 Published Online-Baseline Coverage and Fidelity

The experimental baseline coverage is deliberately separated into (i) a common-protocol comparator that was actually executed in this study and (ii) published online methods whose algorithmic scope is documented but whose original implementations were not rerun under the present Random-Forest/CPJITSDP pipeline. The executed comparator provides a controlled contextual baseline across Tomcat, JGroups, Spring-Integration, and Camel. The published literature additionally provides BORB, ODaSC, PBSA, and online cross-project methods as established reference points [3, 5, 15]. BORB is particularly close to the present setting because it adapts offline learners to online JIT-SDP with verification latency and adaptive resampling. The corrected BORB publication specifies a 90-day waiting period for open-source data, periodic retraining from the accumulated

Table 7: Mean F1 of Temporal Adaptation Strategies Across Four Additional Projects

Project	Cumulative	SW-2000	SW-4000	SW-6000	SW-8000	Adaptive threshold
Camel	0.2831	0.2237	0.2397	0.2498	0.2670	0.3118
JGroups	0.3294	0.2678	0.3043	0.3073	0.3093	0.3135
Spring-Integration	0.6667	0.6472	0.6583	0.6655	0.6667	0.6428
Tomcat	0.4508	0.4250	0.4277	0.4362	0.4478	0.4401

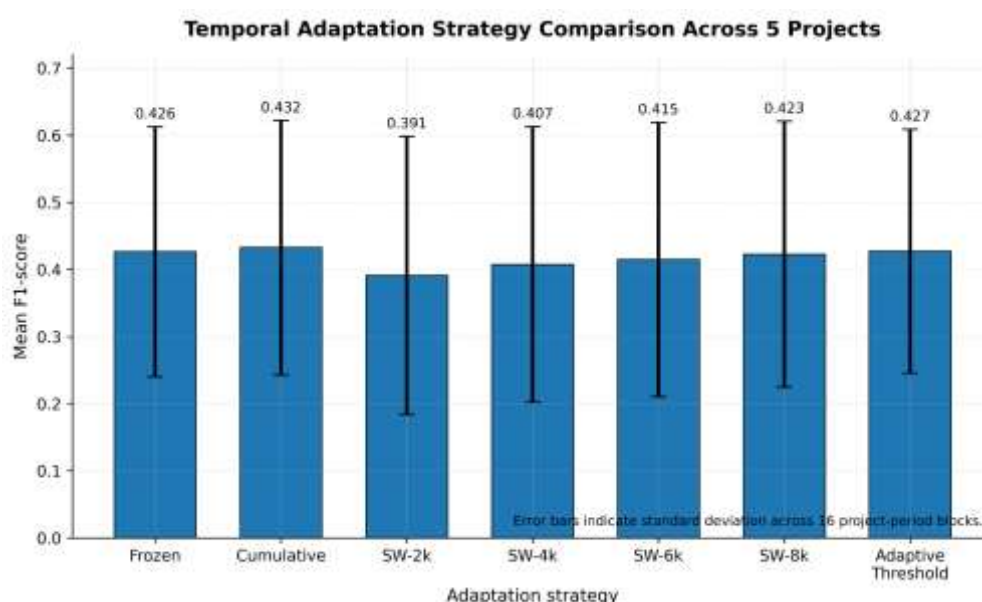


Figure 6: Mean F1-score of the temporal adaptation strategies across the 16 project-period blocks. Error bars indicate standard deviation across the project-period observations. No strategy is interpreted as universally superior.

labeled training set, adaptive prediction thresholds, and oversampling-rate boosting [3]. ODaSC and PBSA address complementary online-learning and drift-reliability issues [5]. Because the available published baselines use different learner

implementations, tuning protocols, and performance measures, their published numerical results are not copied into the present result tables. This avoids presenting cross-paper numbers as if they were matched experiments. The resulting claim is therefore precise: the study contains an executed published-baseline-style comparator and a documented comparison against the established online JIT-SDP baseline families, but it does not claim exact reproduction of every published implementation.

11 Uncertainty and Statistical Analysis

Bootstrap confidence intervals were computed with 2,000 resamples for frozen and adaptive-threshold future-test predictions. The F1 confidence intervals overlap for every project. For example, Tomcat has frozen F1 0.4560 (95% CI 0.4340–0.4782) and adaptive F1 0.4477 (0.4256–0.4696); Spring-Integration has 0.6708 (0.6473–0.6939) and 0.6398 (0.6156–0.6626). For the multi-project adaptation analysis, the 16 project-period observations for each strategy are compared using the Friedman test. The statistic is 31.4018 with $p < 0.0001$ and Kendall's $W = 0.3271$. The significant omnibus result indicates that at least one strategy differs from another; it does not establish that adaptation is superior. Pairwise Wilcoxon comparisons against the frozen strategy yield $p = 0.2744$ for cumulative adaptation, 0.0042 for sliding-2000, 0.0654 for sliding4000, 0.2522 for sliding-6000, 0.8603 for sliding-8000, and 0.8603 for adaptive thresholding. After Holm correction, these comparisons are interpreted conservatively. The significant sliding-2000 comparison is in the direction of lower F1 than the frozen strategy. For the original Brackets-only four-period analysis, the Friedman test was not significant ($p = 0.4669$) and the derived Kendall's W is approximately 0.2342. This difference between the Brackets-only and multi-project statistical analyses illustrates why the multi-project extension is important, while also requiring cautious interpretation because only four temporal units are available per project.

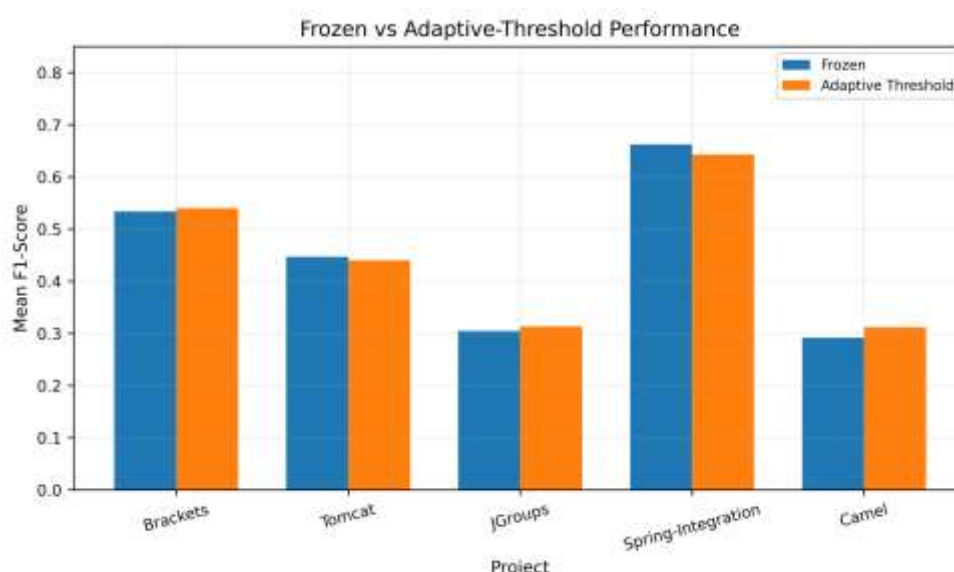


Figure 7: Frozen versus adaptive-threshold performance across the five evaluated projects. The project-dependent differences illustrate why adaptation is treated as a maintenance option rather than a universally superior strategy.

Table 8: Additional Brackets Sensitivity Analyses

Method	F1	ROC-AUC	PR-AUC	MCC
Balanced class weight	0.5366	0.7961	0.5161	0.3900
SMOTE	0.5363	0.7943	0.5122	0.3896
ADASYN	0.5256	0.7958	0.5101	0.3764
XGBoost T1	0.5586	0.7870	0.5751	0.3855
XGBoost T2	0.5044	0.7586	0.5281	0.3413
XGBoost T3	0.4810	0.7481	0.4534	0.3123

XGBoost T4	0.4780	0.7842	0.3960	0.3549
------------	--------	--------	--------	--------

12 DISCUSSION

12.1 RQ1: Feature Selection

RFE provides a compact representation, but its effect is learner-dependent. In the Brackets experiment, the final six-feature Random Forest provides the controlled representation used for temporal analysis. For the additional projects, feature selection is performed separately within historical development data to avoid future leakage.

12.2 RQ2: Distribution Shift

The Brackets feature analysis shows substantial historical-to-future distribution changes, particularly for developer experience and change-size-related variables. These measurements justify the use of future-oriented evaluation but should not be interpreted as direct evidence of concept drift.

12.3 RQ3: Temporal Performance

The multi-project results demonstrate that future performance can change considerably after the initial deployment period. Spring-Integration shows the largest T1-to-T4 F1 reduction, while JGroups is non-monotonic. Therefore, temporal degradation is better characterized as project-dependent instability than as a universal downward trend.

12.4 RQ4: Temporal Adaptation

No adaptation strategy is consistently superior across the four projects. Cumulative and longer sliding windows often remain competitive, while short recency windows can reduce performance. Adaptive thresholding also varies by project and does not consistently outperform a fixed threshold. The results therefore support adaptation as a practical maintenance option rather than as a universally superior solution.

Table 9: Frozen Model Versus Published-Baseline-Style Comparator

Project	Frozen F1	Baseline F1	$\Delta F1$	Frozen AUC	Baseline AUC	ΔMCC
Tomcat	0.4565	0.4646	+0.0081	0.7706	0.7658	+0.0098
JGroups	0.3145	0.3662	+0.0518	0.7456	0.7588	+0.0471
Spring-Integration	0.6741	0.6774	+0.0033	0.8618	0.8576	+0.0055
Camel	0.2975	0.2875	-0.0100	0.7886	0.7727	-0.0227
Mean	0.4356	0.4489	+0.0133	0.7917	0.7887	+0.0099

Table 10: Baseline coverage and experimental fidelity

Baseline family	Role in prior work	Present study	Fidelity statement
BORB	Offline learning adapted to online JIT-SDP	Executed contextual comparator	Implemented with a controlled Random-Forest protocol, but not claimed as exact BORB reproduction.
ODaSC	Confidence-aware online ensemble and adaptive resampling	Literature/positioning baseline	Not numerically rerun under the present learner and stream implementation.

PBSA	Prediction-based concept-drift adaptation under verification latency	Literature/positioning baseline	Not numerically rerun under the present learner and stream implementation.
DeepJIT/CC2Vec	Learned change/code representations	Literature/limitation baseline	Not numerically rerun because their artifact pipelines and datasets are not equivalent to the present five-project protocol.
JITLine	Deployment-faithful JIT-SDP evaluation and simpler deep representation	Literature/positioning baseline	Used to motivate chronology and evaluation-fidelity requirements; no unmatched cross-paper score is reported.

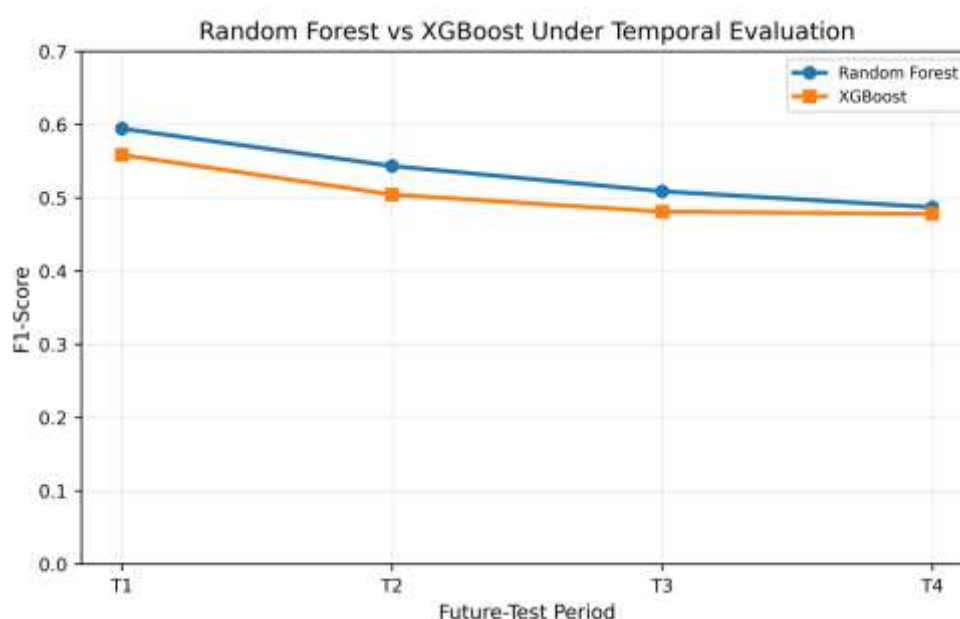


Figure 8: Temporal F1-score comparison between the controlled Random Forest and XGBoost on the Brackets future periods. Both learners exhibit temporal variation, supporting the interpretation that the observed phenomenon is not specific to one tree-based implementation.

13 PRACTICAL IMPLICATIONS

Chronological evaluation should be preferred when the intended use is prediction of future software changes. Model monitoring should consider both predictive metrics and changes in feature distributions. If adaptation is deployed, window size should be treated as a projectspecific operational parameter rather than assumed to have a universally optimal value. False positives can increase developer review burden, whereas false negatives can allow defect-inducing changes to pass through automated prioritization. In addition, verification latency means that labels used for adaptation may arrive after deployment decisions have already been made. Therefore, any production deployment should monitor both predictive performance and label availability, retain human review for consequential decisions, and recalibrate thresholds when the operational cost of errors changes. The present experiments do not evaluate downstream developer productivity, so such benefits should not be inferred from F1 alone. The adaptive-threshold configuration also illustrates an important deployment consideration: threshold calibration and model retraining are coupled. The calibration observations are subsequently included in the final 4,000record fit. Future-test

labels remain isolated, but the reuse means that threshold selection and final fitting are not fully independent and may introduce mild optimism.

14 THREATS TO VALIDITY

14.1 Project and Dataset Coverage

The study now includes five projects, but the five datasets are not statistically representative of all software repositories. Four additional projects come from a processed online-stream collection with explicit delayed-label semantics. Results should therefore be interpreted as evidence across these projects rather than as universal JIT-SDP behavior. 10

14.2 Processed Stream Semantics

The additional CPJITSDP-style datasets contain commit-type fields that encode test events and label availability. Treating them as ordinary independent rows would distort the online protocol; the analysis therefore follows their documented event semantics. The 32 exact duplicate rows present in each additional raw processed stream were removed consistently during construction of the within-project evaluation sets; this preprocessing choice is applied identically across the four additional projects.

14.3 Deep Model Coverage

The study does not reproduce DeepJIT, CC2Vec, or other deep JIT-SDP models under the same multi-project chronological protocol. The DeepJIT and CC2Vec artifact ecosystem is publicly available, but its preprocessing, learned representations, model training requirements, and original benchmark construction differ materially from the CPJITSDP event-stream protocol used here. Their published results are therefore not copied into the present tables because doing so would create a misleading cross-study numerical comparison. The limitation is explicit and the methods are covered in Related Work and Table 10.

14.4 Online-Baseline Fidelity

The study now distinguishes three levels of baseline evidence: an executed contextual online-baseline comparator, documented published online baseline families (BORB, ODaSC and PBSA), and deep-learning baselines (DeepJIT/CC2Vec/JITLine) that are positioned but not numerically reproduced. This is a deliberate methodological choice rather than an omission hidden by the presentation. Exact reproduction would require matching the original implementation, parameter search, stream semantics, and learner stack; the present experiments instead preserve a single common chronological protocol for the models that are actually compared. The distinction is retained in the response-to-reviewers document.

14.5 Temporal Segmentation and Statistical Power

Only four future periods are used per project. This permits interpretable period-wise analysis but limits the number of independent temporal units available for statistical testing. Kendall's W is therefore reported as a descriptive effect size, and statistical conclusions are intentionally conservative.

14.6 Adaptive Threshold Attribution

Adaptive thresholding is coupled to 4,000-record retraining. Consequently, its behavior cannot be attributed exclusively to threshold selection. The fixed-threshold 4,000-record window is reported as the corresponding control.

15 REPRODUCIBILITY AND DATA AVAILABILITY

The complete experimental package is publicly available at <https://github.com/pavanhari6999/temporal-jit-sdp>. It contains the experimental notebook, preprocessing and evaluation procedures, numerical result artifacts, publication and supplementary figures, environment information, and manuscript-related artifacts. The exact Brackets input checksum used in the primary case study is 72d3d69c337a7dd7693363e4 c368c459878ef989be6de3a0e37b8aa1081e81fc. Raw upstream datasets are not redistributed where their source terms do not permit redistribution; dataset provenance and reproduction instructions are provided in the repository. The primary software environment used for the verified experiments includes Python 3.13.5, NumPy 2.3.5, pandas 2.2.3, scikit-learn 1.8.0, XGBoost 3.1.3, SciPy 1.17.0, and Matplotlib 3.10.8. Random seed 42 is used for the primary Random Forest configuration and 2,000 bootstrap resamples are used for confidence intervals.

16 CONCLUSION

This study examined temporal robustness and lightweight adaptation in JIT-SDP under a future-oriented chronological protocol. The original Brackets case study was extended with four additional projects, allowing the analysis to move beyond a single-project evaluation. The experiments show three consistent observations. First, future predictive performance varies substantially across projects. Second, temporal performance can change markedly across future periods, with strong T1-to-T4 degradation in several projects. Third, lightweight adaptation does not provide a uniform improvement: the most effective window differs across projects, and adaptive thresholding is not consistently better than fixed-threshold retraining. The published-baseline-style comparison also produces mixed project-level results rather than a universal advantage for either configuration. Bootstrap intervals overlap between frozen and adaptive models, while the multi-project Friedman analysis detects differences among temporal strategies without establishing one universally superior strategy. The principal contribution is therefore empirical rather than algorithmic. The results demonstrate why JIT-SDP evaluation should respect chronology, monitor temporal distribution changes, and treat adaptation as a project-dependent maintenance decision. The revised study also makes the baseline boundary explicit: an executed online-baseline-style comparator is provided, while BORB, ODaSC, PBSA, DeepJIT, CC2Vec and JITLine are positioned according to their published protocols without presenting unmatched published scores as if they were new experiments. This preserves scientific comparability while making the remaining artifact-level reproduction boundary transparent. 11

Declarations

Author Contributions

D. Pavan Hari Krishna: Conceptualization, methodology, software, data curation, investigation, and writing—original draft. **Sonti Thirupathamma:** Methodology, validation, investigation, visualization, and writing—review and editing. **M. V. D. Prasad:** Supervision, project administration, validation, and writing—review and editing. All authors read and approved the final manuscript.

Funding

No external funding information is reported in the present manuscript.

Conflict of Interest

The authors declare no conflict of interest for this study.

Data Availability

The Brackets dataset is part of the publicly available JITSDP data used in prior research [3, 4]. The additional processed streams are from the CPJITSDP-style project collection used for online JIT-SDP research. Exact input files, checksums, preprocessing decisions, and generated results should accompany the archival replication package.

Code Availability

The replication package, including the experimental notebook, preprocessing and evaluation code, stage result artifacts, environment information, publication figures, supplementary sensitivity results, and manuscript-related artifacts, is publicly available at <https://github.com/pavanhari6999/temporal-jit-sdp>. An archival DOI should be added only after a DOI-bearing release has been created (for example through Zenodo); no DOI is claimed here without such an archival release.

References

- [1] Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, T., 2013. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*. 39(6), 757–773. <https://doi.org/10.1109/TSE.2012.70>
- [2] McIntosh, S., Kamei, Y., 2018. Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. *IEEE Transactions on Software Engineering*. 44(5), 412–428. <https://doi.org/10.1109/TSE.2017.2693980>

- [3] Cabral, G.G., Minku, L.L., Oliveira, A.L.I., Pessoa, D.A., Tabassum, S., 2023. An investigation of online and offline learning models for online just-in-time software defect prediction. *Empirical Software Engineering*. 28, 121. <https://doi.org/10.1007/s10664-023-10335-6>
- [4] Song, L., Minku, L.L., Yao, X., 2023. On the validity of retrospective predictive performance evaluation procedures in just-in-time software defect prediction. *Empirical Software Engineering*. 28, 124. <https://doi.org/10.1007/s10664-023-10341-8>
- [5] Cabral, G.G., Minku, L.L., 2023. Towards reliable online just-in-time software defect prediction. *IEEE Transactions on Software Engineering*. 49(3), 1342–1358. <https://doi.org/10.1109/TSE.2022.3175789>
- [6] Hoang, T.T., Dam, H.K., Kamei, Y., Lo, D., Bissyandé, T.F., 2019. DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction. In: *Proceedings of the 16th International Conference on Mining Software Repositories*, pp. 34–45. <https://doi.org/10.1109/MSR.2019.00016>
- [7] Zeng, Z., Zhao, Y., Zhang, Z., Zhang, T., Xia, X., 2021. Deep just-in-time defect prediction: How far are we? In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. <https://doi.org/10.1145/3460319.3464819>
- [8] Dai, H., Xi, J., Dai, H.-L., 2025. Improve cross-project just-in-time defect prediction with dynamic transfer learning. *Journal of Systems and Software*. 219, 112214. <https://doi.org/10.1016/j.jss.2024.112214>
- [9] Wu, Q., Wang, X., Wei, D., Chen, B., Dang, Q., 2025. Just-in-time software defect prediction method for non-stationary and imbalanced data streams. *Software Quality Journal*. 33, 14. <https://doi.org/10.1007/s11219-025-09711-w>
- [10] Liu, X., Zhou, Y., Tang, Y., Qian, J., Zhou, Y., 2025. Human-in-the-loop online just-in-time software defect prediction: What have we achieved and what do we still miss? *Science of Computer Programming*. 244, 103296. <https://doi.org/10.1016/j.scico.2025.103296>
- [11] Zhao, Y., Damevski, K., Chen, H., 2023. A systematic survey of just-in-time software defect prediction. *ACM Computing Surveys*. 55(10), 201. <https://doi.org/10.1145/3567550>
- [12] Guyon, I., Weston, J., Barnhill, S., Vapnik, V., 2002. Gene selection for cancer classification using support vector machines. *Machine Learning*. 46, 389–422. <https://doi.org/10.1023/A:1012487302797>
- [13] Breiman, L., 2001. Random forests. *Machine Learning*. 45(1), 5–32. <https://doi.org/10.1023/A:1010933404324>
- [14] Jahanshahi, H., Jothimani, D., Bener, A., Cevik, M., 2019. Does chronology matter in JIT defect prediction? A partial replication study. In: *Proceedings of the 15th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 90–99.
- [15] Tabassum, S., Minku, L.L., Feng, D., Cabral, G.G., Song, L., 2020. An investigation of cross-project learning in online just-in-time software defect prediction. In: *Proceedings of the 42nd International Conference on Software Engineering*. <https://doi.org/10.1145/3377811.3380403>
- [16] Pornprasit, C., Tantithamthavorn, C.K., 2021. JITLine: A simpler, better, faster, finer-grained just-in-time defect prediction. In: *Proceedings of the 18th IEEE/ACM International Conference on Mining Software Repositories*, pp. 369–379. <https://doi.org/10.1109/MSR52588.2021.00049>
- [17] Chen, T., Guestrin, C., 2016. XGBoost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. <https://doi.org/10.1145/2939672.2939785>