SPECTRUM OF ENGINEERING SCIENCES
ISSN (E) 3007-3138 (P) 3007-312X VOLUME 3, ISSUE 12, 2025

OPERATIONAL REVIEW-QUEUE CHURN UNDER TEMPORAL RETRAINING IN JUST-IN-TIME SOFTWARE DEFECT PREDICTION

Muhammad Arif Shah

School of Computing, Pak-Austria Fachhochschule Institute of Applied Sciences and Technology

arif.websol@gmail.com

DOI: https://doi.org/10.5281/zenodo.23363320

Keywords
Just-in-time defect prediction; Temporal retraining; Prediction stability; Prediction churn; Risk ranking; Review prioritisation; Software quality assurance; Model maintenance

Article History
Received: 01 November 2025
Accepted: 15 December 2025
Published: 31 December 2025

Copyright @Author
Corresponding Author: *
Muhammad Arif Shah

Abstract
Just-in-time software defect prediction (JIT-SDP) models are periodically retrained as software projects evolve, and replacement models are usually judged by aggregate measures such as ROC-AUC. Teams that act on these models, however, review a ranked subset of incoming commits, so a retrained model can preserve its aggregate accuracy while selecting different commits. This operational consequence of temporal retraining has received limited empirical attention. We study it with a strictly chronological protocol on six open-source projects. Each project is split into eight time-ordered windows; at every step, an OLD model trained on the earlier history and a NEW model trained on one additional window score the same subsequent commits. We compare their probabilities, rankings, and fixed-budget review queues for Logistic Regression (LR), Random Forest (RF), and scikit-learn HistGradientBoostingClassifier (HGB), using retrospectively established defect labels. Across 108 OLD-vs-NEW comparisons, 62.96% showed top-20% review-queue Jaccard overlap below 0.75, and 37.04% combined this with an absolute ROC-AUC change of at most 0.02. In one JGroups case, ROC-AUC moved from 0.6878 to 0.6880 while the top-20% queues overlapped by only 0.427. The pattern appeared in the three discovery projects (44.44% stable-performance/high-churn) and in three independent replication projects (29.63%). Churn was strongly learner-dependent: 88.89% of HGB comparisons were high-churn, compared with 58.33% for RF and 41.67% for LR. Smaller review budgets were less stable, and the phenomenon persisted with six, eight and ten windows. Updates that incorporated two windows of new history were associated with lower queue overlap than one-window updates, and denser scores near the review cutoff were associated with lower overlap in most settings. Under the studied protocol, similar aggregate performance did not guarantee similar commit prioritisation, suggesting that model replacements should also be assessed for review-queue stability.

1 INTRODUCTION

Just-in-time software defect prediction (JIT-SDP) estimates, for each incoming commit, the risk that the change introduces a defect [1, 2]. Because the prediction is made when the change is submitted, its output can be used directly to prioritise code review, testing, or additional inspection while the change is still fresh in the developer's mind [2]. In practice, the usual operational output is not a binary label but a ranking: a team with a limited

review capacity examines the highest-risk commits first, often a fixed fraction of the incoming change stream [2, 3, 4].

Software projects change over time, and the relationship between change metrics and defect-proneness changes with them [5, 6]. JIT-SDP models are therefore expected to be retrained periodically as new labelled history accumulates [6, 7]. When a retrained model replaces its predecessor, the replacement is commonly assessed with aggregate measures such as ROC-AUC or average precision (AP). An aggregate measure summarises how well the model separates defective from clean commits on average. It does not say whether the retrained model selects the same commits for review as the model it replaces.

This distinction matters operationally. Two models can have nearly identical ROC-AUC on the same batch of commits and still assign substantially different rankings to individual commits. If a team reviews the top 20% of commits each period, a model update with unchanged ROC-AUC can still send a different set of commits to reviewers. In machine learning, such instance-level disagreement between successive model versions has been called prediction churn [8], and the related observation that equally accurate models can produce conflicting predictions has been studied as predictive multiplicity [9]. In defect prediction, Bowes et al. showed that classifiers with similar overall performance can detect different defective modules [10]. The temporal analogue, in which successive retrained versions of the same JIT model are compared on the same future commits, has received limited explicit attention.

This paper studies that question empirically. We use a retrospective, strictly chronological retraining protocol on six open-source projects from a public JIT-SDP dataset [11]. Each project is split into eight time-ordered windows. At every retraining step, an OLD model trained on earlier history and a NEW model trained on that history plus one more window score the same, strictly later window of commits. Because both models score identical commits, any difference in their outputs is attributable to the retraining update rather than to a change in the evaluation data. We quantify the difference at three levels: predicted probabilities, risk rankings, and fixed-budget review queues (the top 10%, 20%, and 30% of commits by predicted risk). We relate these measures to the change in aggregate performance between OLD and NEW models.

The study proceeded as a single workflow in three stages. A discovery stage on three projects identified the phenomenon. An independent replication stage on three different projects tested whether it persisted and examined its robustness to the number of temporal windows and the review budget. A mechanism stage pooled all six projects to examine how review-queue churn relates to performance change, to score crowding near the review cutoff, and to the amount of history added in a retraining update.

The study addresses five research questions:

- **RQ1.** How much do commit-level predicted risks and risk rankings change between successive temporally retrained JIT defect models?
- **RQ2.** How stable are fixed-budget review queues across successive model versions?
- **RQ3.** How often does substantial review-queue churn occur despite little or no change in aggregate predictive performance?
- **RQ4.** How do learner family, project, review budget, and temporal segmentation affect operational churn?
- **RQ5.** What mechanisms are associated with operational churn, particularly score crowding near review cutoffs and the amount of newly incorporated training history?

The contributions of this paper are empirical:

- An operationalisation of temporal prediction churn in JIT-SDP at the probability, ranking and fixed-budget review-queue levels, evaluated on identical future commits for each OLD/NEW model pair.
- Evidence from six projects that similar aggregate predictive performance after temporal retraining does not guarantee stable commit prioritisation, including replication on three projects not used for discovery.
- Evidence that operational churn is strongly learner-dependent under the studied configurations, with HGB the most churn-prone,

LR generally the most stable, and RF intermediate and more project-dependent.

• Robustness analysis shows that smaller review budgets were less stable and that the phenomenon persisted under six, eight, and ten chronological windows.

• Mechanism-oriented analyses show that score crowding near the review cutoff and larger retraining increments were associated with lower review-queue overlap.

Section 2 reviews related work. Section 3 describes the study design. Section 4 reports results by research question. Sections 5 and 6 discuss the findings and their practical implications. Section 7 addresses threats to validity, Section 8 describes reproducibility, and Section 9 concludes.

2 Background and Related Work

2.1 Just-in-Time Software Defect Prediction

Mockus and Weiss modelled the risk of individual software changes using change properties such as size, diffusion and developer experience [1]. Kamei et al. formalised just-in-time quality assurance in a large-scale study of open-source and commercial projects, using fourteen change-level metrics grouped into diffusion, size, purpose, history and experience dimensions, and evaluated both classification and effort-aware ranking [2]. These metrics, or close variants, underlie many subsequent JIT-SDP datasets, including the one used here. Hassan showed that the complexity of code changes is informative about future faults [12]. Defect-inducing commits are commonly identified with the SZZ approach, which traces bug-fixing commits back to the changes that introduced the modified lines [13]. Because SZZ labels are imperfect, mislabelled changes can affect both model performance and evaluation conclusions [14]. Kamei et al. later studied JIT models in cross-project settings [15]. The present study is a within-project, temporal study and does not address cross-project transfer.

2.2 Temporal Evolution, Model Ageing and Concept Drift

Defect data are generated by evolving projects. Ekanayake et al. showed that defect prediction quality varies over time and linked this variation to changes in the underlying data, using the notion of concept drift [5]. McIntosh and Kamei conducted a longitudinal case study of JIT-SDP and found that the importance of change properties fluctuates over time and that models trained on older data can lose performance, which suggests that JIT models should be kept up to date [6]. Bangash et al. examined the time-based stability of conclusions in cross-project defect prediction and showed that conclusions can change when temporal ordering is respected [16]. In machine learning, concept drift adaptation addresses learning when the joint distribution of inputs and labels changes over time, including strategies based on periodic or triggered retraining [17].

These studies motivate retraining and mainly ask whether performance degrades or recovers over time. The present study asks a complementary question: when a model is retrained on additional history, how much do its commit-level outputs change relative to the model it replaces, and does that change appear in aggregate performance? Prediction churn between successive model versions is a property of the update, not evidence of concept drift in itself, and we do not attempt to detect or characterise drift.

2.3 Verification Latency and Realistic Temporal Evaluation

In deployed JIT-SDP, a commit's defect label becomes known only when a defect-inducing relationship is discovered, which may take a long time. Cabral et al. called this delay verification latency and studied it together with class imbalance evolution in online JIT-SDP, showing that both affect online learning and evaluation [7]. Tabassum et al. investigated cross-project learning in an online JIT-SDP setting that accounts for waiting time before labels become available [18], and Cabral and Minku proposed procedures for more reliable online JIT-SDP [19]. Song et al. examined whether retrospective evaluation procedures that ignore label-availability timing yield valid performance estimates in JIT-SDP and showed that the choice of procedure can change conclusions [20].

The present study does not reproduce verification latency. It uses retrospectively established final labels and a strictly chronological ordering of training and test windows. This design isolates the stability of successive temporally retrained predictors under chronological software evolution using retrospectively established defect labels; it does not attempt to reproduce the exact timing with which defect labels would become available in an online deployment. The results should therefore be read as properties of successive retrained models under a controlled chronological protocol, not as a simulation of a specific deployment.

2.4 Operational Prioritisation and Review Budgets

JIT-SDP outputs are often used to rank commits for inspection under limited resources. Kamei et al. evaluated effort-aware JIT models by the fraction of defect-inducing changes found when inspecting a fixed fraction of changed code [2]. Yang et al. showed that simple unsupervised ranking models could perform well under effort-aware evaluation [3], and Huang et al. revisited this comparison and highlighted the trade-offs between supervised and unsupervised models in effort-aware settings [4]. These studies treat the ranked list and its cutoff as the operational output of JIT-SDP. We follow the same view but study a different property: whether the set of commits above a fixed review-budget cutoff remains the same when the model is retrained. We use commit-count budgets (top 10%, 20% and 30%) rather than code-size effort, and we do not measure developer effort.

2.5 Prediction Consistency, Disagreement and Churn

Bowes et al. compared defect prediction classifiers and found that classifiers with similar overall performance identify different sets of defective modules [10]. In machine learning, Cormier et al. defined prediction churn as the fraction of differing predictions between successive model versions and proposed methods to reduce it when models are updated [8]. Marx et al. showed that many models with near-identical accuracy can assign conflicting predictions to individual instances, a phenomenon they called predictive multiplicity [9]. Together, these works establish that aggregate performance does not determine instance-level outputs. The present study applies this observation to temporal retraining in JIT-SDP and measures it at the level that matters for review prioritisation: the composition of fixed-budget review queues.

2.6 Research Gap

JIT-SDP research has examined change metrics, labelling, temporal performance stability, verification latency and effort-aware ranking. Work on prediction churn and predictive multiplicity has shown that similar performance can coexist with different instance-level outputs. Comparatively little JIT-SDP work has measured how much successive temporally retrained models disagree on identical future commits, how this disagreement translates into the composition of fixed-budget review queues, how often queue changes occur when aggregate performance barely changes, and how these effects depend on the learner, the review budget, the temporal segmentation and the size of the retraining update. This study addresses that gap with a chronological, replicated design.

3 Study Design

Figure 1 summarises the design. Each project is ordered by commit time and split into chronological windows. At every retraining step, an OLD and a NEW model are trained on nested histories and compared on the same subsequent window.

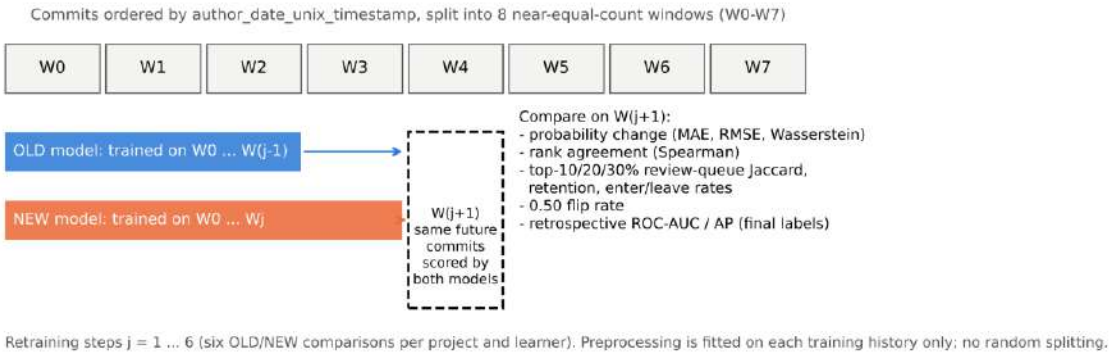


Figure 1. Temporal OLD/NEW retraining design on identical future commits.

3.1 Dataset and Projects

We used the public jit-sdp-data repository, which provides commit-level JIT-SDP data for open-source projects [11]. Projects were eligible if a supported chronological commit timestamp was present. Three projects (Camel, JGroups and Tomcat) formed the discovery set. Seven further projects were screened for independent replication. Four of them (Broadleaf, Nova, npm and Spring Integration) were excluded because the files used in this study did not contain a supported chronological timestamp column, which the protocol requires. The remaining three projects (Brackets, Fabric8 and Neutron) formed the replication set. Table 1 summarises the six projects. They span 118,877 commits, of which 26,336 (22.15%) are labelled defect-inducing, and defect rates range from 0.1728 (JGroups) to 0.2755 (Tomcat).

Table 1. Project characteristics after cleaning.

Project	Role in study	Period (author date)	Commits	Defective	Defect rate	Commits/window
camel	Discovery (Stage 1)	2007-03-19 – 2017-12-07	30,739	6,360	0.2069	≈3,842
jgroups	Discovery (Stage 1)	2003-09-09 – 2017-12-05	18,434	3,185	0.1728	≈2,304
tomcat	Discovery (Stage 1)	2006-03-27 – 2017-12-06	18,960	5,223	0.2755	≈2,370
brackets	Replication (Stage 2)	2011-12-07 – 2017-12-07	17,572	4,143	0.2358	≈2,196
fabric8	Replication (Stage 2)	2011-04-13 – 2017-12-06	13,483	2,736	0.2029	≈1,685
neutron	Replication (Stage 2)	2011-01-01 – 2017-12-27	19,689	4,689	0.2382	≈2,461
Total		–	118,877	26,336	0.2215	–

3.2 Target and JIT Metrics

The target is the dataset's binary label contains_bug, which marks commits labelled as defect-inducing. The predictors are the fourteen JIT change metrics provided by the dataset: fix (whether the commit is a fix), ns (number of modified subsystems), nd (number of modified directories), nf (number of modified files), entrophy (distribution of modified code across files; the dataset's spelling is retained), la (lines added), ld (lines deleted), lt (lines of code in modified files before the change), ndev (number of developers who previously changed the modified files), age (average time since the last change to the modified files), nuc (number of unique prior changes to the modified files), exp (developer experience), rexp (recent developer experience) and sexp (developer experience on the modified subsystems). These metrics follow the definitions of Kamei et al. [2].

3.3 Data Cleaning and Leakage Controls

Commits were ordered by author_date_unix_timestamp. Duplicate commit identifiers were checked, and none were found in any of the six projects. The following fields were excluded from the predictor set because they identify commits or people, contain free text, or are derived from or posterior to the outcome: commit identifiers, author identity and e-mail, commit message, the classification, linked and fixes fields, the glm_probability field supplied with the data, project identity, and any other target-derived or post-outcome field. All preprocessing (median imputation and, for LR, standardisation) was fitted only on the training history of each model and then applied to the test window. Training and test windows never overlapped, and every test window was strictly later than all training data of both models.

3.4 Chronological Temporal Protocol

Each project was split into eight consecutive windows W0–W7 containing near-equal numbers of commits. At step j, for j = 1, ..., 6, the OLD model was trained on W0 to W(j−1) and the NEW model on W0 to Wj; both scored W(j+1). This yields six OLD-vs-NEW comparisons for each project and learner and 108 primary comparisons overall (6 projects × 3 learners × 6 steps). Test windows ranged from 1,685 commits (Fabric8) to 3,843 commits (Camel). Because consecutive steps use different test windows, each commit is evaluated in only one OLD-vs-NEW comparison.
The OLD/NEW design compares two models that differ only in the amount of training history: the NEW model's history contains the OLD model's history plus one additional window. Performance and churn are computed on identical commits, so differences between OLD and NEW outputs reflect the retraining update rather than differences in evaluation data. Defect labels are the final labels in the dataset. As discussed in Section 2.3, this design isolates the stability of successive temporally retrained predictors under chronological software evolution using retrospectively established defect labels; it does not attempt to reproduce the exact timing with which defect labels would become available in an online deployment.

3.5 Learning Algorithms

We used three learner families that are common in defect prediction and differ in inductive bias: Logistic Regression (LR), Random Forest (RF) [21] and gradient-boosted decision trees implemented with scikit-learn's HistGradientBoostingClassifier (HGB) [22]. Configurations were fixed in advance and were not tuned per project or step (Table 2). All learners used random seed 42. Fixing the configuration ensures that differences between OLD and NEW models arise from the training history rather than from hyperparameter search.

Table 2. Learners and fixed model configurations.

Learner	Implementation	Preprocessing	Configuration
LR	LogisticRegression	Median imputation; StandardScaler	solver = liblinear; max_iter = 3000

Learner	Implementation	Preprocessing	Configuration
RF	RandomForestClassifier	Median imputation	n_estimators = 300; min_samples_leaf = 3; max_features = sqrt; n_jobs = −1
HGB	HistGradientBoostingClassifier	Median imputation	learning_rate = 0.08; max_iter = 180; max_leaf_nodes = 31; l2_regularization = 0.1

3.6 Performance Metrics

The primary performance measures are ROC-AUC and AP, computed for both OLD and NEW models on the shared test window. The change in performance is ΔAUC = AUC(NEW) − AUC(OLD), and analogously ΔAP. As secondary measures, we computed the Matthews correlation coefficient (MCC), balanced accuracy, recall, precision and F1 at the conventional 0.50 threshold. The secondary measures are not used for the screening rule because review-budget queues do not depend on a fixed probability threshold.

3.7 Operational Churn Metrics

Table 3 defines the churn measures. Probability churn is measured with the mean absolute error (MAE) and root-mean-square error (RMSE) between OLD and NEW probabilities on the same commits, and with the Wasserstein distance between the two probability distributions. MAE and RMSE capture commit-level change; the Wasserstein distance captures only distributional change and can be small even when individual commits move substantially. Ranking churn is measured with the Spearman correlation between OLD and NEW probabilities.

Operational review-queue churn is measured at fixed review budgets. For a budget of k% (k = 10, 20, 30), the review queue of a model is the set of the k% highest-scored commits in the test window. The Jaccard overlap between the OLD and NEW queues equals 1 when both models select identical commits and 0 when they select disjoint sets. Because both queues have the same size, the fraction of commits entering the queue equals the fraction leaving it, and Jaccard overlap is a monotone function of retention r, the share of the OLD queue that remains in the NEW queue: J = r / (2 − r). A retention of 0.80, for example, corresponds to a Jaccard overlap of about 0.67. The top-20% budget is the primary budget, and the 10% and 30% budgets are used for robustness. Finally, the label-flip rate is the fraction of commits whose predicted label at threshold 0.50 differs between OLD and NEW.

Table 3. Predictive-performance and operational-churn measures.

Group	Measure	Definition / interpretation
Performance	ROC-AUC, AP (primary)	Threshold-free discrimination and average precision of each model on the shared test window; ΔAUC = AUC_new − AUC_old.
Performance	MCC, BA, recall, precision, F1 (secondary)	Classification measures at the conventional 0.50 probability threshold.
Probability churn	MAE, RMSE	Mean absolute and root-mean-square difference between p_old and p_new over the same commits.
Probability churn	Wasserstein distance	Distance between the marginal distributions of p_old and p_new (distributional shift, not commit-level change).

Group	Measure	Definition / interpretation
Ranking churn	Spearman correlation	Rank correlation between p_old and p_new over the test window.
Queue churn	Top-k% Jaccard (k = 10, 20, 30)	|Q_old ∩ Q_new| / |Q_old ∪ Q_new|; 1 means identical review queues.
Queue churn	Retention; enter/leave rate	Fraction of Q_old that remains in Q_new; fraction of the queue that enters (equivalently, for fixed-size queues, leaves).
Decision churn	Label-flip rate at 0.50	Fraction of commits whose predicted label at threshold 0.50 differs between OLD and NEW.
Screening	High churn; stable-performance/high-churn	High churn: top-20% Jaccard < 0.75. Stable-performance/high-churn: |ΔAUC| ≤ 0.02 and top-20% Jaccard < 0.75. Descriptive rules, not significance thresholds.

3.8 Stable-Performance/High-Churn Screening Rule

To summarise how often queue changes coincide with little performance change, we used a descriptive screening rule. A comparison is high-churn if its top-20% Jaccard overlap is below 0.75. With fixed-size queues, J = 0.75 corresponds to a retention of about 0.857, so a high-churn comparison replaces more than roughly one in seven commits of the review queue. A comparison is stable-performance/high-churn if it is high-churn and |ΔAUC| ≤ 0.02. These cut-offs are descriptive screening thresholds chosen to make the phenomenon countable. They are not statistical significance thresholds, and we report sensitivity to them over a 3 × 3 grid (|ΔAUC| ≤ 0.01, 0.02, 0.03; Jaccard < 0.60, 0.75, 0.90) in Appendix A.

3.9 Three-Stage Experimental Strategy

The study was organised as one workflow with three stages. Stage 1 (discovery) applied the protocol to Camel, JGroups and Tomcat, giving 54 primary comparisons. Stage 2 (independent replication and robustness) applied the identical protocol to Brackets, Fabric8 and Neutron, giving another 54 primary comparisons. Stage 2 also repeated the analysis with six and ten windows and reported all three review budgets. Stage 3 (mechanism) pooled the 108 primary comparisons of Stages 1 and 2 without retraining them and added three analyses. The first classified comparisons into performance/churn quadrants. The second related review-queue overlap to the fraction of commits whose scores lie close to the review cutoff (within ±0.01, ±0.02 and ±0.05) and to the score gap at the cutoff. The third compared one-window updates with two-window updates. In the retraining-increment analysis, the smaller update is the primary comparison at steps 2–6, whose OLD model lacks one window of the NEW model's history. The larger update compares the same NEW model with an OLD model trained on a history ending one window earlier, so that two windows of history are added. Both updates are evaluated on the same test window, which gives 30 paired comparisons per learner. The results section is organised by research question, and each question draws on the stages that address it.

3.10 Analysis Stance

All analyses are descriptive. We report means, medians, fractions of comparisons meeting the screening rules and Spearman rank correlations between churn measures and candidate explanatory quantities. We do not perform inferential significance tests, and correlations are interpreted as associations. Reproducibility details are given in Section 8.

4 Results

4.1 RQ1: Change in Predicted Risks and Rankings

Tables 4 and 5 report per-project and per-learner results for the discovery and replication stages, and Table 6 aggregates probability and ranking churn by learner across all six projects.

Table 4. Stage 1 discovery results by project and learner.

Project	Learner	AUC OLD → NEW	Mean |ΔAUC|	Mean Spearman	Top-20% Jaccard	Mean prob. MAE	High churn	SP/HC
camel	HGB	0.790 → 0.801	0.0120	0.911	0.624 (0.541)	0.0838	5/6	4/6
camel	LR	0.679 → 0.712	0.0325	0.917	0.731 (0.351)	0.0813	3/6	1/6
camel	RF	0.792 → 0.802	0.0167	0.915	0.661 (0.555)	0.0656	5/6	3/6
jgroups	HGB	0.734 → 0.741	0.0175	0.620	0.411 (0.260)	0.1002	6/6	4/6
jgroups	LR	0.668 → 0.670	0.0149	0.923	0.756 (0.532)	0.0416	2/6	0/6
jgroups	RF	0.738 → 0.746	0.0125	0.810	0.564 (0.480)	0.0701	6/6	4/6
tomcat	HGB	0.780 → 0.784	0.0040	0.883	0.650 (0.572)	0.0875	5/6	5/6
tomcat	LR	0.631 → 0.665	0.0345	0.895	0.743 (0.409)	0.0815	3/6	0/6
tomcat	RF	0.782 → 0.787	0.0062	0.931	0.745 (0.652)	0.0535	3/6	3/6

Table 5. Stage 2 replication results by project and learner.

Project	Learner	AUC OLD → NEW	Mean |ΔAUC|	Mean Spearman	Top-20% Jaccard	Mean prob. MAE	High churn	SP/HC
brackets	HGB	0.848 → 0.871	0.0231	0.902	0.682 (0.445)	0.0798	6/6	4/6
brackets	LR	0.772 → 0.781	0.0123	0.971	0.842 (0.522)	0.0592	1/6	0/6
brackets	RF	0.870 → 0.876	0.0068	0.977	0.802 (0.753)	0.0419	0/6	0/6
fabric8	HGB	0.788 → 0.826	0.0457	0.798	0.482 (0.271)	0.1149	6/6	2/6

Project	Learner	AUC OLD → NEW	Mean |ΔAUC|	Mean Spearman	Top-20% Jaccard	Mean prob. MAE	High churn	SP/HC
fabric8	LR	0.653 → 0.736	0.1005	0.682	0.581 (0.129)	0.0791	4/6	0/6
fabric8	RF	0.792 → 0.826	0.0339	0.881	0.593 (0.318)	0.0700	6/6	3/6
neutron	HGB	0.913 → 0.918	0.0060	0.948	0.699 (0.624)	0.0700	4/6	4/6
neutron	LR	0.873 → 0.878	0.0049	0.975	0.843 (0.718)	0.0422	2/6	2/6
neutron	RF	0.919 → 0.921	0.0040	0.991	0.788 (0.748)	0.0406	1/6	1/6

Across the 108 comparisons, the mean absolute difference between OLD and NEW probabilities for the same commit was 0.0702, and the mean RMSE was 0.1073 (Table 6). The mean Wasserstein distance (0.0448) was smaller than the mean MAE, which indicates that part of the commit-level change consisted of commits exchanging probability mass rather than a shift of the overall probability distribution. Retraining therefore moved individual commits more than it changed the shape of the score distribution.

Rank correlations between OLD and NEW scores were high on average (mean Spearman 0.885) but varied widely. A third of comparisons (36 of 108) had a Spearman correlation below 0.90, and fewer than half (47 of 108, 43.5%) reached 0.95. The extreme case was Fabric8 with LR at the first step, where the rank correlation was −0.204. That comparison also had the largest performance change (ΔAUC = 0.3276), reflecting a very short initial training history rather than ordinary model maintenance. On average, 7.58% of commits changed predicted label at the 0.50 threshold.

Table 6. Probability and ranking churn by learner across all six projects.

Learner	Prob. MAE	Prob. RMSE	Wasserstein	Spearman	Flip rate (0.50)	Top-20% retention	Top-20% enter rate
HGB	0.0894	0.1499	0.0471	0.844 (0.493)	0.0892	0.7330	0.2670
RF	0.0569	0.0823	0.0338	0.917 (0.738)	0.0593	0.8114	0.1886
LR	0.0642	0.0898	0.0534	0.894 (-0.204)	0.0787	0.8400	0.1600
All	0.0702	0.1073	0.0448	0.885 (-0.204)	0.0758	0.7948	0.2052

Probability and ranking churn differed by learner. HGB had the largest mean probability change (MAE 0.0894, RMSE 0.1499) and the lowest mean rank correlation (0.844). RF had the smallest probability change (MAE 0.0569) and the highest mean rank correlation (0.917). LR had intermediate probability change (MAE 0.0642) and mean rank correlation (0.894), but the widest range: its minimum rank correlation (−0.204) and maximum label-flip rate (0.4538) were the most extreme of any learner. LR also had the largest Wasserstein distance (0.0534), which is consistent with retraining shifting the overall LR score distribution more than it reorders commits.

The discovery and replication stages agree on this pattern. In Stage 1, HGB had the lowest mean rank correlation in each of the three projects, and JGroups with HGB had a mean rank correlation

of only 0.620 (Table 4). In Stage 2, HGB again had the lowest rank correlation in Brackets and Neutron, while in Fabric8 LR had the lowest (0.682), reflecting the large early performance changes of LR in that project (Table 5).

Answer to RQ1. Successive temporally retrained models assigned materially different risks to the same commits: the mean absolute probability change was 0.0702, and a third of comparisons had a rank correlation below 0.90. Change was largest for HGB and smallest for RF at the probability and ranking levels.

4.2 RQ2: Stability of Fixed-Budget Review Queues

Rank correlations summarise the whole ranking, whereas a team reviews only the top of it. Across the six projects, the mean top-20% Jaccard overlap between OLD and NEW queues was 0.6774, and mean retention was 0.7948. On average, therefore, roughly one in five commits in the top-20% queue of the NEW model would not have been selected by the OLD model for the same batch. The mean top-10% and top-30% overlaps were 0.6036 and 0.7150. Overall, 68 of 108 comparisons (62.96%) were high-churn under the screening rule. Rank correlation and top-20% overlap were closely related (Spearman 0.885 between the two measures across comparisons). Even so, high overall rank correlation did not ensure a stable top-20% queue, because small reorderings near the cutoff are enough to exchange commits between the reviewed and unreviewed sets.

Queue stability differed by project (Tables 4 and 5). Mean top-20% overlap was lowest in Fabric8 (0.5520) and JGroups (0.5770) and highest in Neutron (0.7766) and Brackets (0.7751), with Camel (0.6717) and Tomcat (0.7123) in between. The two most stable projects were also the projects in which the NEW models had the highest mean ROC-AUC (0.905 for Neutron and 0.843 for Brackets). Even there, however, HGB queues changed substantially: Brackets with HGB was high-churn in six of six comparisons, and Neutron with HGB in four of six.

Figure 2 shows top-20% overlap by retraining step. Overlap tended to increase in later steps (mean 0.527 at step 1 and 0.754 at step 6, pooled over projects and learners), and the share of high-churn comparisons fell from 94.4% at step 1 to 44.4% at steps 5 and 6. Later steps add one window to a longer history, so each update is proportionally smaller. Step and relative update size are confounded in the primary design, is Section 4.5 examines update size with a paired comparison at fixed NEW models. Even at the last step, almost half of the comparisons remained high-churn.

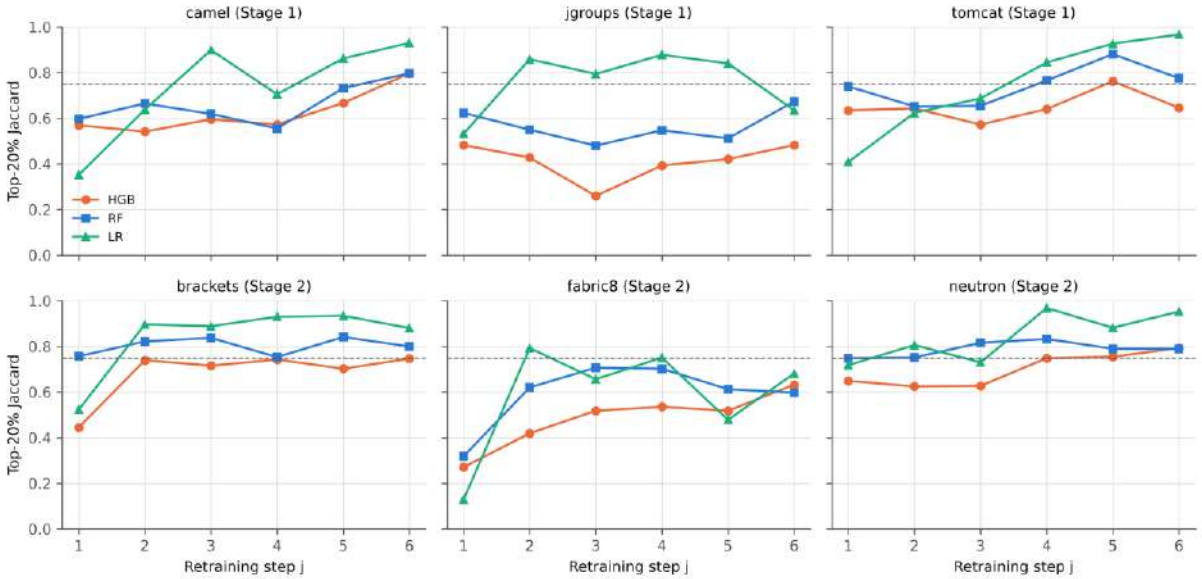


Figure 2. Top-20% review-queue overlaps by retraining step across discovery and replication projects; dashed line = Jaccard 0.75.

Answer to RQ2. Fixed-budget review queues were often unstable across successive model versions. At the top-20% budget, the mean overlap was 0.6774 and 62.96% of comparisons were high-churn. Stability varied by project and tended to improve as the training history grew, but substantial churn persisted at later steps.

4.3 RQ3: Review-Queue Churn without Performance Change

Figure 3 plots |ΔAUC| against top-20% overlap for all 108 comparisons. Most comparisons involved small performance changes: 77 of 108 (71.30%) had |ΔAUC| ≤ 0.02. Of these, 40 (51.95%) were high-churn. Overall, 40 of 108 comparisons (37.04%) were stable-performance/high-churn. In 27 of these 40 cases, the NEW model had a slightly higher ROC-AUC than the OLD model, so a replacement judged by ROC-AUC alone would usually have been accepted as neutral or marginally beneficial.

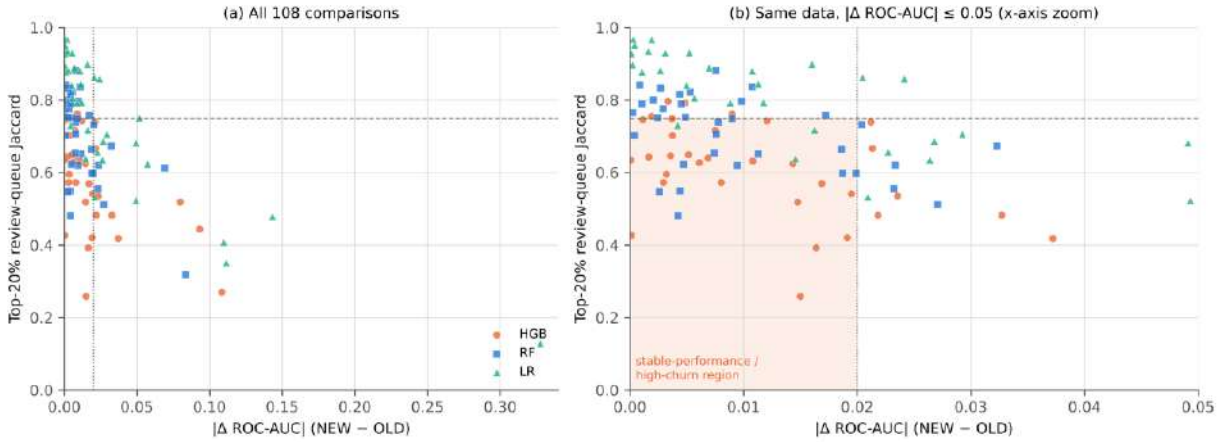


Figure 3. Absolute ROC-AUC change versus top-20% review-queue overlap across all 108 comparisons; shaded region denotes the stable-performance/high-churn screen.

The discovery stage identified the pattern. In Stage 1, 70.37% of comparisons were high-churn and 44.44% were stable-performance/high-churn, with each discovery project contributing the same stable-performance/high-churn share (44.44%). No comparison in Stage 1 had ROC-AUC at or above 0.995, so the pattern was not an artefact of near-perfect models whose rankings are unconstrained by performance. Only 5.56% of Stage 1 comparisons were uniformly stable across the stability criteria used in the screening diagnostics.

The replication stage confirmed the pattern on three independent projects at a lower but substantial level: 55.56% of comparisons were high-churn and 29.63% were stable-performance/high-churn. Each replication project contributed stable-performance/high-churn cases (Brackets 22.22%, Fabric8 27.78%, Neutron 38.89%), and 7.41% of Stage 2 comparisons were uniformly stable. Again, no comparison reached ROC-AUC 0.995.

A JGroups example illustrates the phenomenon. At Stage 1, step 2, the HGB OLD and NEW models scored the same 2,304 commits with ROC-AUC 0.6878 and 0.6880 (ΔAUC = 0.0001). Their top-20% queues overlapped by only 0.427 (retention 0.599), their rank correlation was 0.701, and the mean absolute probability change was 0.133. Judged by ROC-AUC, the update was neutral; judged by the commits sent for review, roughly 40% of the queue changed. A second JGroups case shows that the pattern is not limited to a single step: at step 3, HGB had ΔAUC = −0.0150 and a top-20% overlap of 0.260, the second-lowest overlap among all 108 comparisons. The stable-performance/high-churn fraction depends on the screening thresholds, as expected for a descriptive rule (Table A1). With the Jaccard threshold held at 0.75, tightening the performance criterion to $|\Delta AUC| \leq 0.01$ gave 27.78% (Stage 1) and 18.52% (Stage 2), and relaxing it to 0.03 gave 61.11% and 37.04%. With a stricter churn criterion (Jaccard < 0.60) and $|\Delta AUC| \leq 0.02$, the fractions were 24.07% and 3.70%. The replication projects therefore had fewer extreme queue changes than the discovery projects, but under every combination in the grid some stable-performance/high-churn cases were observed in Stage 1, and in Stage 2 all combinations except the strictest one did so.

Answer to RQ3. Substantial review-queue churn frequently occurred with little or no change in ROC-AUC: 37.04% of all comparisons, and 51.95% of the comparisons with $|\Delta AUC| \leq 0.02$, were stable-performance/high-churn. The pattern was observed in the discovery projects and in the independent replication projects, and its frequency varies with the descriptive thresholds used to count it.

4.4 RQ4: Effects of Learner, Project, Review Budget and Temporal Segmentation

Learner family. Learner family was the most consistent factor. Across the six projects, 88.89% of HGB comparisons were high-churn, compared with 58.33% for RF and 41.67% for LR. The HGB share was the same (88.89%) in Stage 1 and Stage 2, and its mean top-20% overlap was the lowest of the three learners in both stages (0.5912 across six projects). RF was intermediate overall but project-dependent: it was high-churn in every JGroups and Fabric8 comparison but in none of the Brackets comparisons. LR had the highest mean overlap (0.7492) and the lowest high-churn share. However, LR was not uniformly stable: in Fabric8 its minimum top-20% overlap was 0.129, and its stability was associated with larger performance changes when churn did occur (Section 4.5). Under the studied configurations, therefore, the learner ordering was HGB < RF < LR for queue stability, but the size of the gap and the position of RF varied by project.

Project. Project effects were substantial (Tables 4 and 5). Fabric8 and JGroups had the lowest mean top-20% overlap for HGB and RF, and Fabric8 had the lowest for LR, while Brackets and Neutron were the most stable projects for RF and LR. These differences did not track defect rate in an obvious way; for example, JGroups had the lowest defect rate (0.1728) and Fabric8 one of the lower ones (0.2029), yet Tomcat, with the highest defect rate (0.2755), was intermediate.

Review budget. Smaller review budgets were less stable (Table 7, Figure 4). In Stage 2, mean HGB overlap rose from 0.5167 at the top-10% budget to 0.6210 at top-20% and 0.6951 at top-30%. For RF the corresponding values were 0.6167, 0.7273 and 0.7904. Every Stage 2 HGB comparison had a top-10% overlap below 0.75. For LR the gradient was weaker (0.7283, 0.7552 and 0.7854), and across the six projects LR overlap was nearly flat between the top-20% and top-30% budgets (0.7492 and 0.7538). In the Stage 1 projects alone, LR overlap was slightly higher at the top-20% than at the top-30% budget (0.7433 and 0.7222), so the budget gradient is clear for HGB and RF but not monotone for LR in every subset. At the top-10% budget, the median Stage 2 overlap was 0.5520 for HGB, which means that a typical update changed close to half of the highest-priority review set for that learner.

Table 7. Review-budget robustness across learners.

Learner	Budget	Mean Jaccard	Median Jaccard	Share < 0.75	Share < 0.90	Six-project mean Jaccard
HGB	Top-10%	0.5167	0.5520	100.00%	100.00%	0.5023
HGB	Top-20%	0.6210	0.6404	88.89%	100.00%	0.5912
HGB	Top-30%	0.6951	0.7292	61.11%	100.00%	0.6475
RF	Top-10%	0.6167	0.6439	88.89%	100.00%	0.6004
RF	Top-20%	0.7273	0.7547	38.89%	100.00%	0.6919
RF	Top-30%	0.7904	0.8337	27.78%	94.44%	0.7438
LR	Top-10%	0.7283	0.7735	50.00%	61.11%	0.7080
LR	Top-20%	0.7552	0.7992	38.89%	77.78%	0.7492
LR	Top-30%	0.7854	0.8643	27.78%	72.22%	0.7538

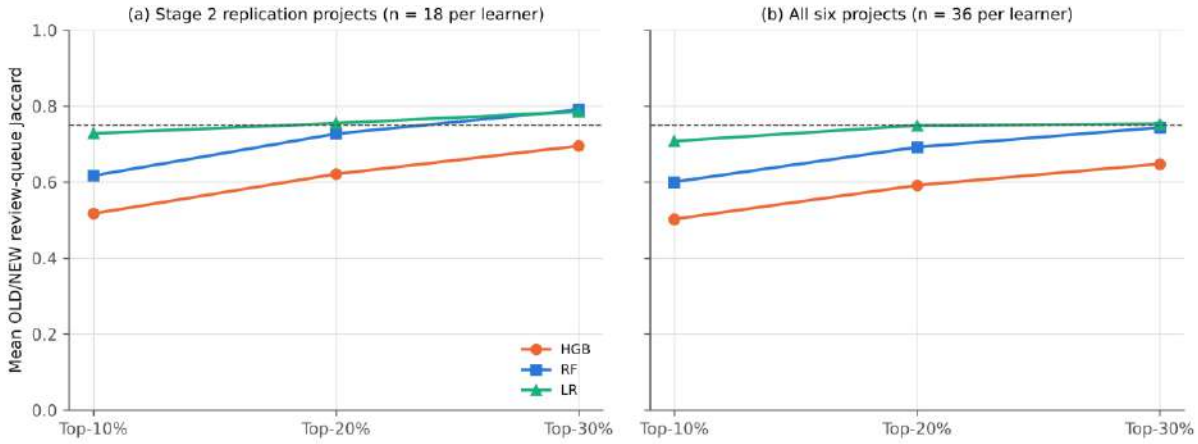


Figure 4. Review-queue Jaccard overlap at top-10%, top-20%, and top-30% budgets.

Temporal segmentation. The phenomenon persisted when the Stage 2 projects were split into six or ten windows instead of eight (Table 8). The high-churn share was 63.89%, 55.56% and 44.93% for six, eight and ten windows, and the stable-performance/high-churn share was 33.33%, 29.63% and 24.64%. Finer segmentation, which adds a smaller fraction of new history at each step, was associated with higher overlap at all three budgets and with higher mean rank correlation. This gradient is consistent with the retraining-increment analysis in Section 4.5. Under every segmentation, about a quarter or more of comparisons were stable-performance/high-churn.

Table 8. Temporal-segmentation robustness in the Stage 2 projects.

Windows	Valid comparisons	High churn	SP/HC	Mean J top-10%	Mean J top-20%	Mean J top-30%	Mean Spearman	Mean |ΔAUC|
6	36	63.89%	33.33%	0.5912	0.6771	0.7346	0.9002	0.0249
8	54	55.56%	29.63%	0.6206	0.7012	0.7570	0.9028	0.0264
10	69	44.93%	24.64%	0.6638	0.7379	0.7974	0.9370	0.0144

Answer to RQ4. Operational churn was strongly learner-dependent under the studied configurations: HGB was the most churn-prone in both stages, LR was generally the most stable, and RF was intermediate and project-dependent. Churn also varied by project, was greater at smaller review budgets for HGB and RF, and persisted under six, eight and ten chronological windows, decreasing as segmentation became finer.

4.5 RQ5: Mechanisms Associated with Operational Churn

Performance change. Table 9 classifies the 108 comparisons into performance/churn quadrants. For LR, high churn was mostly accompanied by performance change: 12 of LR's 15 high-churn comparisons were unstable-performance cases, and the Spearman correlation between |ΔAUC| and top-20% overlap was −0.854 (−0.890 for |ΔAP|). For HGB, the situation was reversed: 23 of its 32 high-churn comparisons (63.89% of all HGB comparisons) occurred with |ΔAUC| ≤ 0.02, and the correlation was weaker (−0.553 for |ΔAUC|, −0.626 for |ΔAP|). RF was intermediate (14 stable-performance/high-churn comparisons; ρ = −0.458 for |ΔAUC| and −0.325 for |ΔAP|). Larger performance changes were thus associated with more churn for every learner, most strongly for LR, but for HGB and RF a large share of churn was not accompanied by a

measurable change in aggregate performance. Only LR had comparisons in the unstable-performance/low-churn quadrant (3 of 36).

Table 9. Stage 3 performance/churn quadrants by learner.

Learner	Stable perf. / high churn	Unstable perf. / high churn	Stable perf. / low churn	Unstable perf. / low churn	ρ(|ΔAUC|, J)	ρ(|ΔAP|, J)
HGB	23/36 (63.89%)	9/36 (25.00%)	4/36 (11.11%)	0/36 (0.00%)	-0.553	-0.626
RF	14/36 (38.89%)	7/36 (19.44%)	15/36 (41.67%)	0/36 (0.00%)	-0.458	-0.325
LR	3/36 (8.33%)	12/36 (33.33%)	18/36 (50.00%)	3/36 (8.33%)	-0.854	-0.890

Score crowding near the review cutoff. If many commits have scores close to the review-budget cutoff, a small change in their scores after retraining can move them across the cutoff. We measured, for each comparison, the fraction of test commits whose NEW score lay within ±ε of the NEW top-k% cutoff score, and related it to Jaccard overlap (Table A2). In 24 of the 27 learner × budget × ε combinations, the correlation was negative: comparisons with more crowding near the cutoff tended to have lower queue overlap. At the top-20% and top-30% budgets, the association was strongest for RF (ρ between −0.638 and −0.769 at top-20%, and between −0.714 and −0.791 at top-30%) and moderate for LR and HGB (ρ between −0.278 and −0.530). At the top-10% budget, it was moderate to strong for LR (ρ between −0.465 and −0.563), weaker for RF (ρ between −0.225 and −0.532), and absent for HGB (ρ between 0.001 and 0.024). Typical crowding was not extreme: at the top-20% budget, the median share of commits within ±0.01 of the cutoff ranged from 1.2% (HGB) to 2.6% (LR), and within ±0.05 from 5.8% to 13.0%.

The score margin at the cutoff was very small. At the top-20% budget, the median NEW-model margin was about 0.0003 for all learners, and the largest top-20% margin in any comparison was below 0.004. A screening diagnostic that required a cutoff margin above 0.01 was therefore met by none of the high-churn comparisons. This outcome reflects the scale of that criterion relative to dense score distributions of several thousand commits rather than the absence of cutoff effects. The margin measures themselves were mostly weakly positively correlated with overlap (Table A2) and are less informative than the crowding fractions.

Two qualifications apply. First, crowding does not explain differences between learners: LR had the densest scores near the cutoff but the highest overlap, and HGB had the least dense but the lowest overlap. The association holds within learners, not across them. Second, the correlations are moderate, and the HGB top-10% results show no association. Score crowding is therefore a plausible contributing mechanism, associated with lower overlap in most settings, but it does not fully explain the stable-performance/high-churn cases.

Retraining increment. Table 10 and Figure 5 compare one-window and two-window updates at the same NEW model and test window. Larger updates were associated with lower overlap for all three learners. Mean top-20% overlap fell from 0.6077 to 0.5587 for HGB, from 0.7042 to 0.6554 for RF, and from 0.8104 to 0.6532 for LR. The larger update produced more churn in 73.33% of paired comparisons for HGB and RF and in 93.33% for LR. Mean probability MAE increased correspondingly, most sharply for LR (0.0408 to 0.1043). LR, the most stable learner under one-window updates, thus showed the largest sensitivity to update size, and its two-window overlap was comparable to that of RF. The

direction agrees with the segmentation analysis (Table 8) and with the step trend (Figure 3).

Table 10. Retraining-increment sensitivity for one-window versus two-window updates.

Learner	n	Mean J smaller	Mean J larger	Mean Δ J (larger − smaller)	Median Δ J	Larger update more churn	Mean prob. MAE smaller → larger
HGB	30	0.6077	0.5587	-0.0490	-0.0313	73.33%	0.0808 → 0.1054
RF	30	0.7042	0.6554	-0.0488	-0.0403	73.33%	0.0535 → 0.0712
LR	30	0.8104	0.6532	-0.1572	-0.1431	93.33%	0.0408 → 0.1043

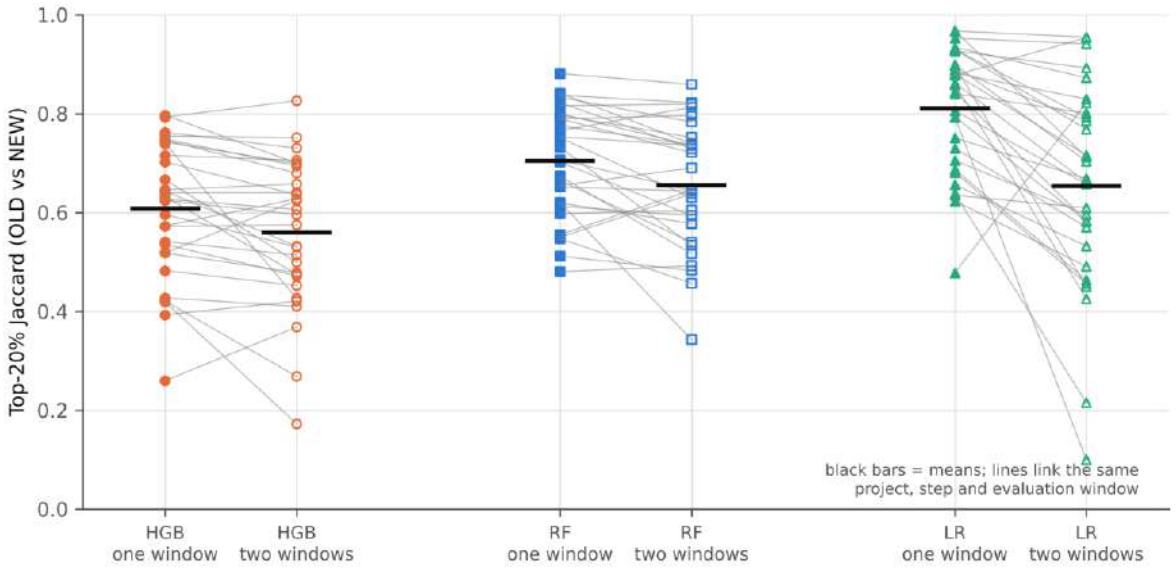


Figure 5. Top-20% review-queue overlap for one-window versus two-window retraining updates.

Answer to RQ5. Three factors were associated with operational churn. Performance change was associated with churn, strongly for LR and more weakly for HGB and RF, whose churn often occurred without performance change. Score crowding near the review cutoff was associated with lower overlap within learners in most budget settings, but not for HGB at the top-10% budget. Larger retraining increments were associated with lower overlap for all learners. These are associations under the studied protocol, not established causal mechanisms.

4.6 Summary of Results

Under a strictly chronological retraining protocol, successive JIT models frequently selected different commits for review even when their ROC-AUC barely changed. The pattern appeared in the discovery projects, replicated in independent projects, and was strongest for HGB and at small review budgets. It weakened but persisted as the temporal segmentation became finer, and it was stronger when updates incorporated more new history.

5. Discussion

Why aggregate metrics can hide operational instability. ROC-AUC measures the probability that a randomly chosen defective commit is ranked above a randomly chosen clean commit. It is invariant to many reorderings that matter operationally: two models can place different defective commits near the top of the ranking, or exchange commits of the same class across the review cutoff, while the global proportion of correctly ordered pairs stays almost the same. A fixed review budget depends on a small portion of the ranking, namely the commits around and above the cutoff. In the studied projects, this portion changed more than the global ranking. The mean rank correlation of 0.885 coexisted with a mean top-20% overlap of 0.677. The JGroups example shows the extreme form: a ROC-AUC change of 0.0001 accompanied a 40% replacement of the review queue. These observations agree with general findings on prediction churn and predictive multiplicity, which show that models with near-identical aggregate performance can disagree substantially on individual instances [8, 9], and with Bowes et al.'s finding that comparable defect classifiers detect different defects [10]. Our results show that the same holds for successive versions of a single JIT model under temporal retraining.

Model-family differences. HGB was the most churn-prone learner in both stages, and most of its churn occurred without measurable performance change. A plausible interpretation is that boosted trees fitted with a fixed number of iterations construct different partitions of the feature space when the training history changes, which reorders commits within regions of similar risk. RF averages many trees and was more stable at the probability level, although it was still high-churn in more than half of its comparisons. LR is a smoother model whose coefficients changed gradually in most steps, but it reacted strongly when the update was large or the early history was short (Fabric8). These interpretations are not tested in this study. They are also specific to the fixed configurations used. Tuned configurations, different numbers of boosting iterations or regularisation choices could change the ordering, and the results do not imply that any learner is universally more or less stable.

Implications for JIT model maintenance. Retraining is generally recommended for JIT-SDP because the relationship between change properties and defects evolves [5, 6]. Our results do not contradict this recommendation, and the study does not evaluate whether retraining improves or harms prediction quality over time. They indicate instead that each retraining is also an operational change: it alters which commits are prioritised, and this change is only partly visible in aggregate performance. Model maintenance decisions may therefore benefit from considering stability alongside accuracy, especially when the model's output directly determines which commits receive additional review.

Review-budget sensitivity. Smaller budgets were less stable for HGB and RF. This follows partly from the structure of the measure: a small queue contains fewer commits, so the exchange of a given number of commits near the cutoff changes a larger share of it. Small budgets are also the setting in which a JIT model is most selective and the reviewed commits are those considered highest-risk. For LR the budget gradient was weak and not consistently monotone. Teams that use small review budgets should therefore expect queue composition to be more sensitive to model updates, at least for tree ensembles under configurations like those studied.

Distinction from concept drift. Concept drift refers to changes over time in the joint distribution of inputs and labels [17]. Prediction churn, as measured here, is a difference between two models evaluated on the same data. It can arise when the NEW training window reflects a drifted distribution, but it can also arise without drift, for example from the variance of a flexible learner fitted to a larger sample or from small score changes near a dense cutoff. The study does not separate these sources and does not interpret churn as evidence of drift. Likewise, the study does not test whether churn predicts future performance degradation.

Relation to model monitoring. Model monitoring in deployment typically tracks input distributions and, when labels become available,

performance. Review-queue overlap between the current and the candidate model can be computed without labels, on the same recent batch of commits, before the candidate is deployed. This differs from verification-latency-aware evaluation [7, 20], which concerns when labels become available to measure performance. Because overlap does not require labels, it could complement label-based monitoring as a measure of the operational impact of an update. The present study does not evaluate such a monitoring procedure; it only shows that overlap carries information that ROC-AUC does not.

Interpretation of score crowding. The crowding results suggest a simple mechanism: when many commits have nearly equal scores around the cutoff, small score perturbations after retraining move commits in and out of the queue. This mechanism is consistent with the within-learner associations but does not explain between-learner differences, because LR had the densest scores near the cutoff and the highest overlap. Crowding thus amplifies the effect of score changes but does not determine their size. A complete explanation would also need to account for how much each learner's scores move after retraining, which differed markedly between learners (Table 6).

Why update size matters. Both the segmentation analysis and the paired increment analysis showed that adding more new history at once was associated with more churn. This matches the intuition that a model retrained on a substantially different training set will differ more from its predecessor. It also suggests that the retraining schedule is an operational parameter: frequent small updates and infrequent large updates may have similar average performance yet differ in how much they change the review queue at each update. The study varied update size only between one and two windows, and update size is related to the age and length of the training history. These results therefore describe an association within that range, not a general rule for choosing retraining frequency.

6 Practical Implications

The following suggestions follow from the evidence in the studied projects and protocol. They are suggestions for teams that use JIT-SDP to prioritise review, not universal requirements.

- **Assess model replacements beyond aggregate performance.** A candidate model with the same or slightly higher ROC-AUC or AP than the current model may still change a substantial share of the review queue. In 27 of the 40 stable-performance/high-churn comparisons, the NEW model had marginally higher ROC-AUC.
- **Compare old and new models on a shared recent batch before deployment.** Scoring the same unlabelled commits with both models and computing top-k% overlap, retention and rank correlation requires no labels and directly quantifies the operational change an update would cause.
- **Monitor review-queue overlap at the budget actually used.** Overlap was lower at smaller budgets for HGB and RF, so stability should be assessed at the operational budget rather than with a global rank measure alone.
- **Consider learner stability when choosing a model family.** Under the studied configurations, HGB produced more queue churn than RF and LR, often without performance change. Where queue stability matters, it may be weighed against other model properties.
- **Treat large historical updates as potentially more disruptive.** Updates that incorporated more new history were associated with greater churn. Teams that retrain infrequently may wish to inspect such updates more closely or communicate changes in prioritisation to reviewers.

These measures describe changes in which commits are prioritised. They do not measure developer effort, review outcomes or the cost of churn, which depend on how a team uses the queue. Whether a changed queue is better or worse can only be judged with labels and workflow data that this study does not include.

7 Threats to Validity

7.1 Construct Validity

Queue stability is operationalised with Jaccard overlap at commit-count budgets of 10%, 20% and 30%. Real review processes may use effort-based budgets (for example, lines of code), dynamic budgets, or thresholds rather than fixed

fractions, and they may combine model output with other information. Jaccard overlap treats all queue changes equally regardless of whether exchanged commits are defective. The high-churn and stable-performance/high-churn rules are descriptive screening thresholds, not significance thresholds, and Appendix A shows how the counts vary with them. Defect labels come from the dataset and are subject to the known limitations of SZZ-style labelling [14].

7.2 Internal Validity

The design uses retrospectively established final labels and does not reproduce verification latency. In an online deployment, the most recent training commits would have incomplete or provisional labels [7, 20], which could change both the models and their churn. The study isolates the effect of chronological retraining on successive predictors but does not estimate the churn of a specific deployment process. Model configurations were fixed and untuned, so the results characterise these configurations, not the learner families in general. Scores near the review cutoff are dense, and ties or near-ties can affect queue membership; the very small cutoff gaps observed suggest that the exact handling of near-equal scores can matter at the margin. Strict chronological ordering prevents training on future commits, but it does not remove other historical data-quality issues such as labelling noise. Finally, step and relative update size are confounded in the primary design; the paired increment analysis addresses this only for one- versus two-window updates. The two-window comparison is available only from step 2 onwards, so it covers 30 rather than 36 comparisons per learner.

7.3 External Validity

The study covers six open-source projects from one public dataset family [11]. Four candidate replication projects were excluded because the required chronological timestamp was not available in the files used, so the replication set is smaller than the screened set. The projects are mature open-source systems written mainly in Java, JavaScript and Python. Results may differ in industrial settings, in projects with different commit practices, with other defect-labelling pipelines, with other metric sets, or with deep learning models of code changes, which this study does not evaluate.

7.4 Conclusion Validity

No inferential significance tests were performed, and no confidence intervals are reported. The evidence consists of descriptive statistics, replication across independent projects and sensitivity analyses over windows, budgets and screening thresholds. The comparisons within a project and learner are not independent, because they share training history and project characteristics. The correlations reported in Section 4.5 are associations over 36 comparisons per learner and budget, or 30 for the increment analysis; they should not be interpreted as causal effects. Heterogeneity between projects and learners was substantial, and pooled fractions should be read together with the per-project and per-learner results.

7.5 Reproducibility

All models used fixed configurations and random seed 42, the data are public, and the chronological protocol is fully specified. Implementation details of library versions or hardware could cause small numerical differences for tree ensembles. All intermediate results, including dataset audits, per-comparison metrics and the boundary and increment analyses, were exported to files to allow independent verification.

8 Reproducibility

The datasets are publicly available from the jit-sdp-data repository [11]. The included projects (Camel, JGroups, Tomcat, Brackets, Fabric8 and Neutron) and the excluded projects (Broadleaf, Nova, npm and Spring Integration, excluded because a supported chronological timestamp was absent) are listed in Section 3.1. Commits were ordered by author_date_unix_timestamp, the target was contains_bug, and the predictors were the fourteen JIT metrics listed in Section 3.2. The temporal protocol used eight near-equal chronological windows, with six and ten windows as robustness settings. Learner configurations are

given in Table 2, and all learners used random seed 42 and scikit-learn implementations [22].

The replication package contains the dataset audits (row counts, defect counts, defect rates, time spans and exclusion reasons), the per-comparison results for all stages, the project/learner summaries, the threshold-sensitivity grids, the window-count and review-budget robustness summaries, the Stage 3 boundary-margin, performance/churn and retraining-increment analyses, and the screening diagnostics.

9 Conclusion

This study examined what happens to commit-level predictions when a JIT defect prediction model is retrained on additional chronological history. Across six open-source projects, successive OLD and NEW models evaluated on the same future commits often prioritised substantially different commits for review, even when their ROC-AUC barely changed. The pattern was identified on three projects and replicated on three others. It was strongly learner-dependent, more pronounced at smaller review budgets, persistent under alternative temporal segmentations, and generally stronger when larger amounts of history were incorporated in an update. Score crowding near the review cutoff was associated with lower queue overlap in most settings, although it did not fully explain the cases of churn without performance change. Under the studied protocol, the same aggregate performance did not mean the same operational decisions. When a JIT model is retrained and its output is used to select commits for review, the stability of the review queue merits assessment alongside its predictive performance.

Declarations

Use of generative AI. Generative AI tools were used to assist with language editing, manuscript structuring, and drafting support. All research questions, methodological decisions, data processing, experiments, numerical results, interpretations, and references were reviewed and verified by the author, who takes full responsibility for the content of the manuscript.

Data availability. The datasets analysed in this study are publicly available from the jit-sdp-data repository [11].

REFERENCES

[1] A. Mockus and D. M. Weiss, "Predicting risk of software changes," Bell Labs Technical Journal, vol. 5, no. 2, pp. 169–180, 2000, doi: 10.1002/bltj.2229.

[2] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," IEEE Transactions on Software Engineering, vol. 39, no. 6, pp. 757–773, 2013, doi: 10.1109/TSE.2012.70.

[3] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in Proc. 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), 2016, pp. 157–168, doi: 10.1145/2950290.2950353.

[4] Q. Huang, X. Xia, and D. Lo, "Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction," Empirical Software Engineering, vol. 24, no. 5, pp. 2823–2862, 2019, doi: 10.1007/s10664-018-9661-2.

[5] J. Ekanayake, J. Tappolet, H. C. Gall, and A. Bernstein, "Time variance and defect prediction in software projects: Towards an exploitation of periods of stability and change as well as a notion of concept drift in software projects," Empirical Software Engineering, vol. 17, no. 4–5, pp. 348–389, 2012, doi: 10.1007/s10664-011-9180-x.

[6] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction," IEEE Transactions on Software Engineering, vol. 44, no. 5, pp. 412–428, 2018, doi: 10.1109/TSE.2017.2693980.

[7] G. G. Cabral, L. L. Minku, E. Shihab, and S. Mujahid, "Class imbalance evolution and verification latency in just-in-time software defect prediction," in Proc. 41st International Conference on Software Engineering (ICSE), 2019, pp. 666–676, doi: 10.1109/ICSE.2019.00076.

[8] Q. Cormier, M. Milani Fard, K. Canini, and M. R. Gupta, "Launch and iterate: Reducing prediction churn," in Advances in Neural Information Processing Systems 29 (NeurIPS), 2016.

[9] C. Marx, F. Calmon, and B. Ustun, "Predictive multiplicity in classification," in Proc. 37th International Conference on Machine Learning (ICML), PMLR, vol. 119, 2020, pp. 6765–6774.

[10] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: Do different classifiers find the same defects?," Software Quality Journal, vol. 26, no. 2, pp. 525–552, 2018, doi: 10.1007/s11219-016-9353-3.

[11] jit-sdp-data: Just-in-time software defect prediction datasets, GitHub repository, https://github.com/dinaldoap/jit-sdp-data (repository archived 1 December 2023).

[12] A. E. Hassan, "Predicting faults using the complexity of code changes," in Proc. 31st International Conference on Software Engineering (ICSE), 2009, pp. 78–88, doi: 10.1109/ICSE.2009.5070510.

[13] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," in Proc. International Workshop on Mining Software Repositories (MSR), 2005, pp. 1–5, doi: 10.1145/1083142.1083147.

[14] Y. Fan, X. Xia, D. A. da Costa, D. Lo, A. E. Hassan, and S. Li, "The impact of mislabeled changes by SZZ on just-in-time defect prediction," IEEE Transactions on Software Engineering, vol. 47, no. 8, pp. 1559–1586, 2021, doi: 10.1109/TSE.2019.2929761.

[15] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," Empirical Software Engineering, vol. 21, no. 5, pp. 2072–2106, 2016, doi: 10.1007/s10664-015-9400-x.

[16] A. A. Bangash, H. Sahar, A. Hindle, and K. Ali, "On the time-based conclusion stability of cross-project defect prediction models," Empirical Software Engineering, vol. 25, no. 6, pp. 5047–5083, 2020, doi: 10.1007/s10664-020-09878-9.

[17] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1–37, 2014, doi: 10.1145/2523813.

[18] S. Tabassum, L. L. Minku, D. Feng, G. G. Cabral, and L. Song, "An investigation of cross-project learning in online just-in-time software defect prediction," in Proc. 42nd International Conference on Software Engineering (ICSE), 2020, pp. 554–565, doi: 10.1145/3377811.3380403.

[19] G. G. Cabral and L. L. Minku, "Towards reliable online just-in-time software defect prediction," IEEE Transactions on Software Engineering, vol. 49, no. 3, pp. 1342–1358, 2023, doi: 10.1109/TSE.2022.3175789.

[20] L. Song, L. L. Minku, and X. Yao, "On the validity of retrospective predictive performance evaluation procedures in just-in-time software defect prediction," Empirical Software Engineering, vol. 28, no. 5, Art. no. 124, 2023, doi: 10.1007/s10664-023-10341-8.

[21] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[22] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

Appendix A. Supplementary Tables

Table A1. Sensitivity of stable-performance/high-churn rates to screening thresholds.

|ΔAUC| threshold	Top-20% J < 0.60	Top-20% J < 0.75	Top-20% J < 0.90
≤ 0.01	12.96% / 0.00%	27.78% / 18.52%	42.59% / 48.15%
≤ 0.02	24.07% / 3.70%	44.44% / 29.63%	64.81% / 64.81%
≤ 0.03	31.48% / 5.56%	61.11% / 37.04%	85.19% / 72.22%

Table A2. Correlations of boundary and margin measures with review-queue overlap.

Budget	Learner	Boundary ε = 0.01	Boundary ε = 0.02	Boundary ε = 0.05	OLD margin	NEW margin
Top-10%	HGB	0.001	0.017	0.024	0.079	0.209
Top-10%	RF	-0.225	-0.400	-0.532	0.161	0.370
Top-10%	LR	-0.465	-0.559	-0.563	0.082	0.284
Top-20%	HGB	-0.344	-0.293	-0.341	0.413	0.086
Top-20%	RF	-0.638	-0.751	-0.769	0.197	0.087
Top-20%	LR	-0.327	-0.278	-0.307	0.255	0.255
Top-30%	HGB	-0.454	-0.420	-0.413	0.237	-0.048
Top-30%	RF	-0.714	-0.734	-0.791	0.501	0.350
Top-30%	LR	-0.530	-0.492	-0.413	-0.198	0.337

Per-comparison results for all 108 primary comparisons, the six- and ten-window comparisons, and the 90 retraining-increment comparisons are provided in the replication package rather than in the manuscript.

IEER
Institute for Excellence in Education & Research