Science of Computer Programming 244 (2025) 103296

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Human-in-the-loop online just-in-time software defect prediction: What have we achieved and what do we still miss?

Xutong Liu [a], Yufei Zhou [b], Yutian Tang [c], Junyan Qian [b], Yuming Zhou [a,*]

[a] National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, Jiangsu, China
[b] Key Lab of Education Blockchain and Intelligent Technology, Ministry of Education, Guangxi Normal University, Guilin, 541004, Guangxi, China
[c] School of Computing Science, University of Glasgow, Glasgow, G12 8QN, Scotland, United Kingdom

ARTICLE INFO

Keywords:
Software change
Defect prediction
Human-in-the-loop
Just-in-time

ABSTRACT

Background. The Online Just-In-Time Software Defect Prediction (O-JIT-SDP) employs an online model to predict whether a new software change will introduce a bug. Previous studies have neglected to consider the interaction between Software Quality Assurance (SQA) personnel and the model, potentially missing opportunities to refine prediction accuracy through human feedback. **Problem.** A recent study introduced the first Human-In-The-Loop (HITL) O-JIT-SDP framework called HumLa, integrating SQA staff feedback without accounting for inspection time to boost the prediction performance of O-JIT-SDP. However, upon a thorough revisit of HumLa, we find that while certain aspects of the HITL O-JIT-SDP system appear feasible in ideal conditions, they prove impractical in real-world context. **Objective.** We aim to reformulate HITL O-JIT-SDP, which are crucial yet absent for practical application. **Method.** We propose four crucial enhancements to facilitate practical application of HITL O-JIT-SDP. First, we advocate for the use of observed labels rather than ground-truth labels to evaluate online classifiers in real-world settings. Second, we suggest refraining from utilizing the entire data stream for normalizing features of each new instance, as was done in HumLa. Third, we propose incorporating non-zero SQA inspection time into the formulation of HITL O-JIT-SDP. Fourth, we introduce real-time statistical classifier comparison into the HITL system. **Result.** Our replication uncovers that the performance evaluation of HumLa under a practical scenario significantly deviate from the originally reported performance under an ideal experimental scenario, potentially diminishing the promise of HITL O-JIT-SDP. Furthermore, with our enhanced HITL O-JIT-SDP framework, we revisit a fundamental question in O-JIT-SDP: the benefits of HITL integration. Our experimental findings demonstrate that HITL not only enhances O-JIT-SDP when SQA feedback surpasses Bug-Fixing Commit (BFC) feedback (by providing training commits with superior label quality in less time) but also improves O-JIT-SDP even when SQA feedback delay equals that of BFC feedback (by consistently delivering training commits with improved label quality). The real-time statistical analysis reveals that HITL approaches generally outperform non-HITL O-JIT-SDP approaches with a statistically significant margin. **Conclusion.** Our work bolsters model evaluation credibility and holds the potential to substantially enhance the value of HITL O-JIT-SDP for industrial applications.

* Corresponding author.
E-mail addresses: xryu@smail.nju.edu.cn (X. Liu), zyf@stu.gxnu.edu.cn (Y. Zhou), Yutian.Tang@glasgow.ac.uk (Y. Tang), qianjunyan@gxnu.edu.cn (J. Qian), zhouyuming@nju.edu.cn (Y. Zhou).

https://doi.org/10.1016/j.scico.2025.103296
Received 14 July 2024; Received in revised form 22 February 2025; Accepted 26 February 2025

Available online 3 March 2025

1. Introduction

Just-In-Time Software Defect Prediction (JIT-SDP) predicts whether a commit is likely to introduce a bug or not by analyzing distributions of historical software changes [1,2]. Tan et al. [3] categorized JIT SDP approaches into cross-validation classification, time-sensitive classification, and online classification. Cross-validation classification unrealistically uses future data and mislabels changes, leading to overly optimistic results; time-Sensitive classification improves on this but suffers from dataset dependence and outdated training data; online classification addresses these issues by incrementally and dynamically updating training sets, ensuring more accurate and robust predictions [3].

Recent studies have treated JIT-SDP as an online learning (also known as a stream learning) problem [4–7], which means that the model is incrementally tested and updated as new instances become available. Specifically, every time a new commit is incoming, it is first used as a test instance to obtain its predicted label. Then, after a certain waiting time, its observed label will be obtained. Based on the predicted and observed labels, the prediction performance of the online classifier (an instance of an online classification algorithm) is evaluated. After that, the commit is used as a training instance to update the online classifier.

Researchers have explored various aspects of the above "test-then-train" O-JIT-SDP, including the "class imbalance evolution" phenomenon [5], the impact of cross-project training data on model performance [7], the challenge and mitigation of concept drift [4], the continuous evaluation procedure for O-JIT-SDP models in practice [6], cross-project O-JIT-SDP [8], and using offline base learners in an online JIT-SDP scenario [9]. Undoubtedly, these studies have deepened our understanding of O-JIT-SDP and enhanced their potential for industrial applications in the future. However, an important problem exists in O-JIT-SDP: how to ensure the reliability of the observed labels. Currently, "waiting-time window + Bug-Fixing Commit (BFC)" is the mainstream approach to acquiring the observed labels. Specifically, if a BFC f is committed to fix a previous commit u within the waiting-time window, then u will be labeled as "bug-inducing" at the time that f is committed. Otherwise, u will be labeled as "clean" when the waiting time is exceeded. Such a labeling mechanism is inherently associated with noise in the observed labels. Considering a BFC that fixes commit u arrives after the waiting time window, then a positive (i.e., actually defective) commit u will be wrongly labeled as negative (i.e., clean). In a recent study [6], the default waiting-time window for BFC is set at 15 days. The reason why the waiting-time window cannot be set as long as we want to decrease the label noise is that a training instance may have been outdated after a very long waiting time and hence is not suitable for updating the online classifier (known as "concept drift" [5]). Considering the fact that it can take months or even years for a bug-inducing commit to be labeled as bug-inducing by a BFC [5] (known as verification latency), this raises a serious concern about the quality of the observed labels, which may affect the correct evaluation and actual performance of O-JIT-SDP.

Recently, a novel human labeling method for O-JIT-SDP, HumLa [10], has been proposed to tackle the above issue. In real development, when SQA staff use online JIT-SDP, their commits are the inputs of the prediction system. Note that SQA during the development phase is typically carried out by developers and the SQA staff we refer to are usually the developers themselves. That is to say, the SQA staff we refer to should have sufficient knowledge to inspect software currently under development. For each given input commit, an online classifier will predict whether it is a bug-inducing commit. For those commits predicted as bug-inducing, SQA staff will inspect them and determine whether they are bug-inducing or not. Clearly, such information can bolster the dependability of observed labels associated with commits. Therefore, HumLa integrates label feedback from SQA staff into the prediction loop to enhance the accuracy of the prediction process. For instances predicted as defective, HumLa receives the observed labels right after SQA staff's instant feedback. Conversely, for instances predicted as clean, HumLa obtains the observed labels in the same way with non-HITL O-JIT-SDP, aka, "waiting-time window + BFC". In this way, the interaction between SQA staff and the online JIT-SDP system can be utilized to enhance the reliability of the observed labels of commits.

Upon thorough examination of the HumLa code, we have identified four implementation setups that are viable under ideal conditions but undermine the practical applicability of the HITL system. First, instead of utilizing observed labels, ground-truth labels of new instances are used to evaluate the current performance of the online classifier in HumLa, thereby skewing the realism of the approach. Second, the entire data stream is utilized to normalize features of each new instance in HumLa, posing logistical challenges that do not align with real-world data processing constraints. Third, in practical scenarios, human code inspection demands time, yet HumLa does not incorporate SQA inspection time into its model. Fourth, HumLa repetitively runs the same online classification algorithm to obtain multiple results for statistical tests, which is unfeasible for online classifiers lacking randomness and real-time monitoring requirements. Our replication of the experiment exposes the **significant deviation from the originally reported performance under ideal experimental scenario**, potentially undermining the efficacy of HITL O-JIT-SDP. Furthermore, we re-examine a fundamental inquiry in O-JIT-SDP, the benefits of HITL integration, with our augmented HITL O-JIT-SDP framework. The experimental analysis conducted across ten projects underscores that the integration of HITL significantly amplifies the effectiveness of O-JIT-SDP, demonstrating a statistically significant improvement.

In summary, our contributions can be outlined as follows:

- Enhancing the initial HITL O-JIT-SDP formulation for practical application by addressing its limitations. First, we conduct a thorough analysis of the pitfalls within the existing framework. Then, to ensure the framework is more applicable to real-world scenarios, we suggest four important enhancements: utilize observed labels for model evaluation to better reflect the nuances of practical use cases; shift from global data to historical data for feature normalization to better capture the temporal dynamics and context-specific characteristics of the data; account for the time required for SQA inspections to align with the operational constraints of industrial processes; and introduce a statistical approach to compare online classifiers in real-time, providing

a robust mechanism for selecting the most effective model during operations. These enhancements significantly bolster the framework's robustness and adaptability in industrial settings, thereby elevating its overall utility for practical applications.

- Undertaking a comprehensive assessment on the efficacy, practical feasibility, and benefits of the enhanced HITL O-JIT-SDP framework. Based on open-source projects, on the one hand, we investigate the validity of observed labels in accurately reflecting model prediction performance, the impact of unrealistic normalization on model prediction performance, the suitable combinations of waiting times to achieve relatively optimal results, and the optimal statistical testing for real-time comparison of online classifiers. On the other hand, we apply the enhanced framework to rigorously re-evaluate the tangible benefits of integrating HITL within the O-JIT-SDP context. Our experimental findings offer profound insights into the nuances of the HITL O-JIT-SDP framework. This not only addresses a gap in experimental data within this critical domain but also provides invaluable insights to steer and inspire future research initiatives in the field.
- Offering a replication kit encompassing the implementation of the enhanced HITL O-JIT-SDP framework, inclusive of associated datasets and scripts [11]. This implementation is available for use, replication, and improvement by both practitioners and researchers. Our tool is built as an extension of the widely-used online learning platform MOA (Massive Online Analysis) [12]. We also release our extended MOA [13], facilitating seamless and adaptable integration into various tasks.

The structure of this paper is as follows. Section 2 overviews the fundamental concepts. Section 3 describes our formulation of the HITL O-JIT-SDP model. Section 4 elaborates on our study design, while Section 5 presents our experimental results. The discussion of our findings is presented in Section 6. Section 7 provides an analysis of related studies. Finally, in Section 8, we conclude the paper and outline directions for future work.

2. Preliminaries

In this section, we introduce the fundamental concepts, including the "test-then-training" approach in online learning, the ground-truth labels and observed labels in O-JIT-SDP, and the merits of HITL O-JIT-SDP.

2.1. "Test-then-training" online learning

"Test-then-training" is a commonly used online learning approach (also known as "prequential" online learning) [14]. In this approach, a newly arrived instance is first used as a test instance to evaluate the performance of the current classifier and then used as a training instance to incrementally update the online classifier. This approach is particularly useful in situations where new data arrives continuously and the classifier needs to be updated on the fly to adapt to the changing data distribution.

During "test-then-training" online learning, the online classifier is updated continuously with the arrival of new instances. Therefore, forgetting mechanisms are commonly used when calculating the performance of classifiers to reflect their recent performances. The most commonly used forgetting mechanisms are sliding windows and fading factors [15,16]. The sliding window mechanism calculates the performance of a classifier with a sliding window overtime of the most recent test instances:

$$M_{\mathrm{t}} = \frac{\sum_{i=t-w+1}^{t} x_i}{w},$$

where:

- M_{t}: The performance metric (e.g., accuracy) calculated within the sliding window at the current time step.
- x_i: The performance of the classifier on the i-th test instance.
- w: The size of the sliding window, i.e., the number of most recent test instances considered.
- t: The current time step.

For example, suppose the sliding window size is $w = 5$ and the classifier predicts the following outcomes for the last five instances: $x_{16} = 1, x_{17} = 1, x_{18} = 0, x_{19} = 1, x_{20} = 1$. The sliding window performance at time $t = 20$ would be:

$$M_{\mathrm{window}} = \frac{1+1+0+1+1}{5} = 0.8 \text{ (80\% accuracy)}.$$

Its disadvantage is the requirement of additional memory to store the historical data. The size of the sliding window directly affects the amount of memory required, which can become a problem when dealing with large amounts of data. In contrast, the fading factor mechanism updates the performance of a classifier by a fading factor α. Specifically, the performance of a classifier at time i with fading factor α is as follows [16]:

$$M_i = \frac{S_\alpha(i)}{N_\alpha(i)},$$

where:

$$S_\alpha(i) = x_i + \alpha \cdot S_\alpha(i-1),$$
$$N_\alpha(i) = 1 + \alpha \cdot N_\alpha(i-1),$$

Here:

- $S_\alpha(i)$ is the fading sum of observations at time i.
- $N_\alpha(i)$ is the corresponding fading increment.
- x_i is the performance of the classifier on the i-th test instance.
- The initial values are $S_\alpha(1) = x_1$ and $N_\alpha(1) = 1$.

A larger value of α gives more weight to past performance, resulting in a smoother performance curve. For example, assume that the fading factor $\alpha = 0.9$ and the classifier has the following performance values on the first five instances: $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$, and $x_5 = 0$.

The calculation for M_5 is as follows:

$$S_\alpha(1) = x_1 = 1, \quad N_\alpha(1) = 1$$

$$S_\alpha(2) = x_2 + \alpha \cdot S_\alpha(1) = 0 + 0.9 \cdot 1 = 0.9, \quad N_\alpha(2) = 1 + 0.9 \cdot 1 = 1.9$$

$$S_\alpha(3) = x_3 + \alpha \cdot S_\alpha(2) = 1 + 0.9 \cdot 0.9 = 1.81, \quad N_\alpha(3) = 1 + 0.9 \cdot 1.9 = 2.71$$

$$S_\alpha(4) = x_4 + \alpha \cdot S_\alpha(3) = 1 + 0.9 \cdot 1.81 = 2.629, \quad N_\alpha(4) = 1 + 0.9 \cdot 2.71 = 3.439$$

$$S_\alpha(5) = x_5 + \alpha \cdot S_\alpha(4) = 0 + 0.9 \cdot 2.629 = 2.3661, \quad N_\alpha(5) = 1 + 0.9 \cdot 3.439 = 4.0951$$

Finally, the performance at time $i = 5$ is:

$$M_5 = \frac{S_\alpha(5)}{N_\alpha(5)} = \frac{2.3661}{4.0951} \approx 0.577 \text{ (57.7\% accuracy).}$$

This example demonstrates how the fading factor mechanism works with $\alpha = 0.9$, giving significant weight to past performance and resulting in a smoother performance curve. As can be seen, compared with the sliding window mechanism, fading factor mechanism is memoryless and can adapt to changes in the data stream without storing the historical data. Therefore, in the literature, it is advisable to calculate evaluation metrics using fading factors [16].

In recent O-JIT-SDP studies [4,9,5,6,8,7], the "test-then-training" paradigm with fading factors was used to evaluate model performance (the fading factor was often $\alpha = 0.99$). In our study, we follow these studies to adopt the same approach to update and evaluate classifiers.

2.2. Ground-truth and observed labels

Assigning labels to commits is a crucial aspect of O-JIT-SDP tasks. In an ideal scenario, accessing the "ground truth" labels of commits would provide an invaluable resource, allowing the online classifier to undergo training and testing processes devoid of any label noise. However, in practice, there is no labeling method that can determine the actual "ground truth" labels of software changes.

As a compromise, the labels used in O-JIT-SDP are known as "observed labels." These labels are obtained from available information from the investigated project, such as BFC [5]. Current O-JIT-SDP studies use an approach called "waiting-time window + BFC" to generate observed labels for commits. Specifically, a waiting time window W is set for new commits, for a commit u produced at time U, if a bug-fixing commit f appears at T and $T < W + U$, and this bug-fixing commit f fixes software changes in commit u, then u is labeled as bug-inducing at time T, otherwise, u is labeled as clean at $W + U$.

3. Human-in-the-loop O-JIT-SDP: workflow, pitfalls, and enhancements

In this section, we first draw our formulation of HITL O-JIT-SDP; then, we analyze pitfalls of state-of-the-art HITL O-JIT-SDP framework, HumLa; last, we propose enhancements for practical application of HITL O-JIT-SDP.

3.1. Workflow of HITL O-JIT-SDP

In the domain of artificial intelligence, the concept of "Human-in-the-loop (HITL)" refers to AI systems where humans engage in continuous interaction with the machine to train, monitor, and update models, particularly when deploying AI models in real-world scenarios. The application of HITL spans various stages of the lifecycle of a learning model [17], encompassing tasks such as dataset annotation [18], edge case handling [19,20], model training [21,17], model predictions inspection [22], outlier monitoring [23], and model explanation [24]. In this section, we first introduce the high-level architecture of HITL O-JIT-SDP, then we describe the involved streaming data in the HITL O-JIT-SDP system, finally, we give our core implementation of HITL O-JIT-SDP system.

High-level architecture. Fig. 1 shows the architecture of HITL O-JIT-SDP, where human involvement is denoted by the blue elements. Whenever a new software change is committed, the online classifier generates a prediction using the features of the commit. If the commit is predicted as a bug-inducing change, it is forwarded to SQA staff for inspection, allowing humans to determine its observed label. Conversely, if the prediction designates the commit as clean, the observed label is assigned after by the "waiting time window + BFC" methodology. Once the observed label is obtained, it is used to evaluate this classifier and subsequently

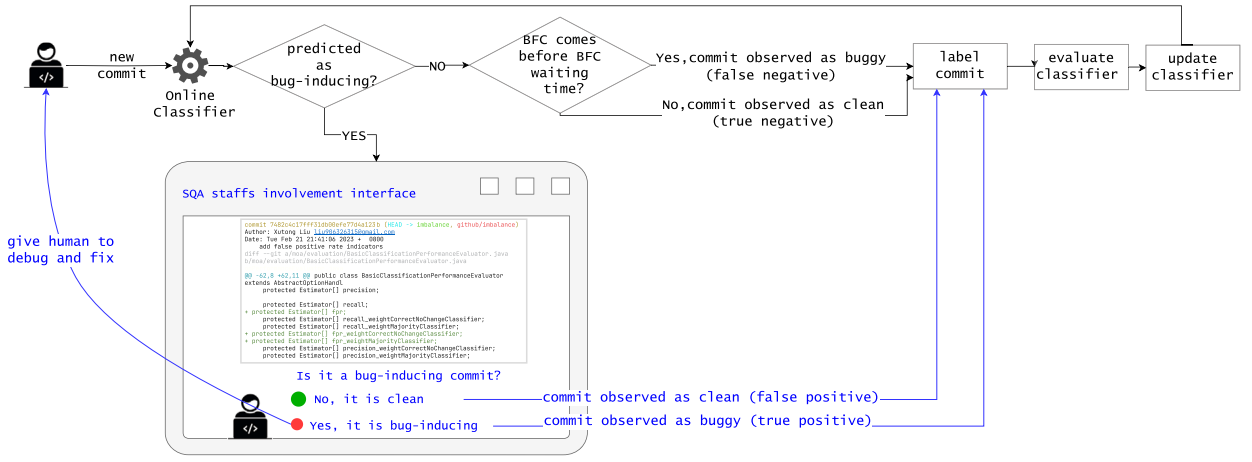


Fig. 1. The high-level architecture of HITL O-JIT-SDP. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

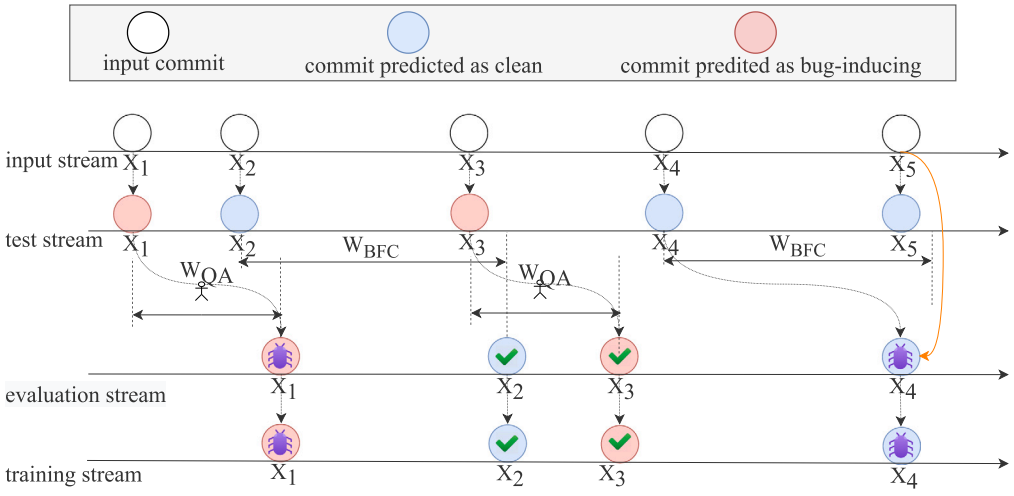


Fig. 2. Streaming data in HITL O-JIT-SDP.

update this classifier accordingly. HITL O-JIT-SDP models enable immediate labeling by SQA staff for commits predicted as positive. Consequently, it improves O-JIT-SDP from two aspects. First, the time delay in updating classifiers with labeled commits is reduced since part of commits are labeled by quick SQA staff's feedback, mitigating the issue of concept drift. Second, since SQA staff possess extensive project-specific knowledge, their judgment regarding the defectiveness of software changes is considered reliable. Therefore, the observed labels provided by them are expected to be less noisy compared to those obtained through non-HITL O-JIT-SDP based on "waiting time window + BFC".

Involved streaming data. Building upon the idea of HITL illustrated in Fig. 1, we further introduce how streaming data flows in such a HITL O-JIT-SDP system. Fig. 2 elucidates how streaming data are produced and employed with time throughout the HITL O-JIT-SDP process, and highlights the generation and utilization of true positive (red commit with bug icon), false positive (red commit with checkmark icon), true negative (blue commit with checkmark icon), and false negative instances (blue commit with bug icon) within the system.

1. *From input stream to test stream.* Each time a commit emerges in the input stream, it is sent to the test stream immediately and then predicted by the current online classifier to obtain its predicted label.
2. *From test stream to evaluation stream.* In the test stream, according to the predicted label of commits, each commit waits in test stream until it is labeled by SQA staff or by "waiting time window + BFC". Specifically, commits predicted as bug-inducing (represented by red, e.g., X_1 and X_3) are sent to SQA staff for code inspection to obtain its observed label while commits predicted as clean (represented by blue, such as X_2 and X_4) wait for their observed labels determined via the "waiting time window + BFC" method. W_{QA} denotes the length of time SQA staff take for code inspection to give the observed label of a commit. As a result, the commit will be either confirmed as bug-inducing (true positive, e.g., X_1) or clean (false positive, e.g., X_3). W_{BFC} denotes the length of time for a commit waiting for its bug-inducing commit. If a BFC that resolves the corresponding

issue appears before the waiting time W_{BFC} elapses (e.g., X_4), it will be labeled as bug-inducing (false negative). If no BFCs that fix the commit appear until the waiting time W_{BFC} (e.g., X_2) passes, it will be labeled as clean (true negative). A commit is sent from test stream to evaluation stream right after it obtains its observed label.

3. *From evaluation stream to training stream.* After arriving at evaluation stream, a commit is immediately used to evaluate the current performance of the online classifier using its observed and predicted labels. After that, the commit is sent to the training stream for updating the current online using its features and observed label.

Algorithm 1 Human-in-the-loop Online Just-In-Time Software Defect Prediction (HITL O-JIT-SDP).

```
1: Initialize input stream, test stream, positive queue, negative queue, evaluation stream, and training stream
2: for each new_commit in input_stream do
3:     test_stream.append(new_commit)
4: end for
5: for each current_commit in test_stream do
6:     Ts_current = current_commit.Ts
7:     current_commit.predicted_label = online_classifier.predict(current_commit.features)
8:     if current_commit.predicted_label is “bug-inducing" then
9:         positive_queue.append(current_commit)
10:    else
11:        negative_queue.append(current_commit)
12:    end if
13:    for each commit in positive_queue do
14:        Ts_commit = commit.Ts
15:        if W_QA < Ts_current − Ts_commit then
16:            commit.observed_label = getSQAInspectionResult(commit)
17:            positive_queue.pop(commit)
18:            online_classifier.evaluate(commit.observed_label, commit.predicted_label)
19:            online_classifier.update(commit.observed_label, commit.features)
20:        end if
21:    end for
22:    for each commit in negative_queue do
23:        Ts_commit = commit.Ts
24:        if W_BFC < Ts_current − Ts_commit then
25:            commit.observed_label = “clean"
26:            negative_queue.pop(commit)
27:        else if current_commit is bug-fixing commit for commit then
28:            commit.observed_label = “bug-inducing"
29:            negative_queue.pop(commit)
30:        end if
31:        online_classifier.evaluate(commit.observed_label, commit.predicted_label)
32:        online_classifier.update(commit.observed_label, commit.features)
33:    end for
34: end for
```

Core process. Algorithm 1 describes the core process of the above HITL system in detail. Initially, each time a commit emerges in the input stream, it is appended to the test stream (Lines 2 to 4). Subsequently, the current online classifier assesses each commit to obtain its predicted label (Lines 5-7). Following this assessment, the commit is placed in a waiting queue (*positiveQueue* or *negativeQueue*) to await its observed label: commits predicted as bug-inducing are queued in the *positiveQueue* queue (Line9); whereas commits predicted as clean are placed in the *negativeQueue* queue (Line 11). Then, commits in both queues receive their observed labels through distinct strategies. After obtaining the observed labels, these commits will be sent to the evaluation stream to incrementally update the classifier's performance using their observed labels and predicted labels. After being evaluated, the commits are incrementally used to update the online classifier using their features and observed labels.

1. **Commits in *positiveQueue* (Lines 13 to 21).** For each commit in positiveQueue, it will be popped and sent to SQA staff for code inspection. We assume SQA staff predict labels of commits with high accuracy since they have prior knowledge of the project.

Following HumLa [10], we set the default error ratio of SQA staff as 0%. According to [10], the O-JIT-SDP with SQA feedback performs better than without SQA feedback when the human error ratio is less than 50%, which means a loose requirement of human accuracy. The length of time it takes for the inspection process to obtain the observed label is set as W_{QA}. As a result, the commit will be either confirmed as bug-inducing or as clean in Line 16. After that, the commit is sent to the evaluation stream to evaluate the current performance of the online classifier (Line 18), and then sent to the training stream to update the online classifier (Line 19).

2. **Commits in** *negativeQueue* **(Lines 22 to 33).** For each commit in *negativeQueue*, if a BFC that resolves the corresponding issue appears before the waiting time W_{BFC} elapses, it will be popped from *negativeQueue* and labeled as bug-inducing (false negative) as in Line 28. If no BFCs that fix the commit appear until the waiting time W_{BFC} passes, it will be popped from *negativeQueue* and labeled as clean (true negative) as in Line 26. After that, the commit is sent to the evaluation stream to evaluate the current performance of the online classifier (Line 31), and then sent to the training stream to update the online classifier (Line 32).

3.2. Pitfalls of state-of-the-art HITL O-JIT-SDP

3.2.1. An implementation of HITL O-JIT-SDP: HumLa

HumLa [10], as an implementation of the aforementioned HITL O-JIT-SDP system, can be delineated across several key dimensions. First, it disregards the time expended by SQA personnel for inspection, effectively setting the W_{QA} parameter to zero. Second, it establishes a default waiting time of 15 days (W_{BFC}) for labeling commits predicted as negative by BFC. Third, HumLa operates under the assumption of a 100% human inspection rate, implying that all commits predicted as positive are routed for human inspection and labeling by default. The investigation into the effect of human inspection rates on HITL system performance reveals a consistent augmentation, irrespective of the specific ratio employed. Fourth, HumLa assumes a 0% human error rate, presuming perfect accuracy in commit labeling by SQA personnel. The examination of human error rates on HITL system efficacy underscores that as long as the human error rate remains below 50%, HITL significantly bolsters the performance of O-JIT-SDP.

The original investigation of HumLa reveals a noteworthy enhancement in the average G-mean, rising from 0.620 to 0.639 subsequent to the integration of HITL into O-JIT-SDP. As affirmed in [10], HumLa "... can significantly benefit predictive performance of JIT-SDP when the human label quality and quantity are above a given threshold." In our forthcoming study, we intend to execute HumLa using its original replication kit while meticulously scrutinizing its source code. This scrutiny aims to delve into potential pitfalls, the practical efficacy of HumLa, and most importantly, we will reassess the impact of HITL in O-JIT-SDP once we rectify any implementation issues in HumLa and introduce an updated HITL O-JIT-SDP system.

3.2.2. Impractical setups in the inital implementation

After thoroughly inspecting the source code of HumLa [10], we have identified four aspects that are viable in the ideal experimental environment but impractical in the real world. As a consequence, the practical deployment of the HITL O-JIT-SDP system may be hindered.

Employment of inaccessible instance labels. HumLa uses the ground truth labels of commits to evaluate online classifiers rather than observed labels. As can be seen in Fig. 1, after the observed label of a commit is obtained, the predicted and observed labels are used to evaluate the current online classifier. However, in the implementation of HumLa [10], the predicted and ground truth labels are used to evaluate the current online classifier, which is viable in the ideal experimental environment but unrealistic during the real-world O-JIT-SDP process. In this study, we clarify the usage of predicted labels and observed labels under the whole HITL O-JIT-SDP process in Algorithm 1. Furthermore, we investigate the validity of using observed label for performance evaluation on reflecting the ground truth performance of HumLa in Section 5.1 as our first RQ (Research Question). Specifically, we want to answer: to what degree does the performance observed from accessible real-world labels accurately reflect the true performance based on ground truth labels (which are available in an ideal experimental setting but not in real-world scenarios)?

Utilization of unachievable instance normalization. When building an online classifier, HumLa uses feature normalization to eliminate the effect of quantitative features measured on different scales. In practice, feature normalization should be restricted to utilizing data observed up to the evaluation point. As can be seen in Fig. 1, under online learning, when a new commit arrives, the feature distribution of instances that will arrive after it is invisible for the current normalization. However, HumLa normalizes new-coming instances based on the feature distribution of all instances in the datastream, including those instances that have not appeared yet, presenting a departure from practicality. This implementation issue not only makes HumLa unrealistic in practical deployment but may result in normalized features inaccurately representing current patterns and distributions, potentially leading to misleading model performance assessments. Hence, in this study, we propose a modification wherein the normalization scaler is solely computed based on previous commits for the current commit. Additionally, we thoroughly investigate the potential bias introduced by this approach in HumLa's performance evaluation in Section 5.2.

Ignorance of SQA inspection time. In Section 3.1, HITL O-JIT-SDP formulation requires the setting of two crucial parameters: W_{BFC} and W_{QA}. W_{BFC} denotes the duration we allow for a BFC commit to determine the label of a commit predicted as non-defective. Conversely, W_{QA} signifies the timeframe allocated to SQA staff for code inspection to ascertain the label of a commit predicted as defective. Prior research has commonly adopted a default value of 15 days for W_{BFC} [6], a convention we maintain in our current investigation. However, the preceding HITL O-JIT-SDP model [10], HumLa, disregards W_{QA} by setting it to 0. Yet, in practical scenarios, human code inspection demands time, and the inspection period W_{QA} may vary for each commit. When we decide the W_{QA} for each commit, two factors should be taken into consideration. On the one hand, a longer W_{QA} can increase the possibility of concept drift affecting the classifier, rendering the commit unsuitable for updating the classifier. On the other hand,

to ensure the acquisition of dependable observed labels, SQA staff need sufficient time to inspect an incoming commit. Therefore, a tradeoff exists between long W_{QA} and short W_{QA}. In Section 3.3.1, we integrate W_{QA} into the HITL O-JIT-SDP framework and delve into realistic configurations for SQA inspection time. Subsequently, in Section 5.3, we investigate optimal default values of W_{QA} to augment the practical utility of the HITL system.

Absence of real-time statistical monitoring provision. When comparing two different O-JIT-SDP models over the same commit data stream, it becomes imperative to perform rigorous statistical testing to discern any performance disparities at specific time points. In past O-JIT-SDP studies, several works [4,6,25,7] have adopted the Wilcoxon signed rank test as a statistical test to study whether the difference in performance between the test and control groups is significant. However, the previous approach has limitations and therefore is insufficient to support real-time statistical performance evaluation of O-JIT-SDP. First, existing statistical comparisons among O-JIT-SDP classifiers depend on the repetitive running of a classifier on the same stream data to generate multiple observations (i.e., performances of a classifier) for a treatment (i.e., a compared classifier). However, repetitive runs are unable to generate multiple observations for non-randomness algorithms, such as the Hoeffding Tree [26], which is a commonly used algorithm in online learning. Therefore, this statistical comparison approach is unfeasible for online classifiers lacking randomness. Second, the efficiency of repetitive running to obtain multiple observations is low and real-time monitoring requirements cannot be satisfied under this approach since the observations required by the statistical test can only be obtained after repeating stream learning several times. In Section 3.3.2, we offer streamlined real-time statistical analysis to conduct a dependable real-time comparison for O-JIT-SDP classifiers. Moreover, in Section 5.4, we select appropriate statistical methods for this real-time statistical analysis framework under the context of O-JIT-SDP.

3.3. Crucial enhancements for practical application

In addressing the obstacles to the practical deployment of the HITL O-JIT-SDP system outlined above, the initial two can be rectified by modifying certain lines of code in the original HumLa implementation. Conversely, the latter two require substantial enhancements to both the foundational structure of the HITL O-JIT-SDP system and its associated evaluation framework. In this section, we elaborate on our proposed enhancements to address the issues of neglecting SQA inspection time and lacking real-time statistical monitoring capabilities.

3.3.1. Incorporating the time required for SQA inspections

Our HITL O-JIT-SDP formulation takes W_{QA} into consideration and allows customized W_{QA} values. In practice, estimating W_{QA} is not straightforward, as it depends on various hard-to-quantify environmental and individual factors. For simplicity, we set it as a constant in our experiments, acknowledging that its accurate assessment involves challenges. To determine a suitable default value for W_{QA}, we examined the typical development frequency and debug frequency in ten open-source projects (detailed in Section 4.2 to estimate the average W_{QA} required for SQA staff in HITL O-JIT-SDP systems. These ten open-source projects have been active for at least two years. On average, these projects witness 9.1 commits per day and 2.8 BFCs per day, indicating a frequent occurrence of day-to-day development and code review in these active projects. Based on this analysis, we assume that commits sent to SQA staff can be inspected daily when a project is under active development and maintenance. Consequently, we set the default value of W_{QA} to seven days, allowing for a large tolerance in the time consumed by code inspection. Additionally, the impact of the W_{QA} value on the performance of O-JIT-SDP will be discussed in Section 5.3.

3.3.2. Offering streamlined real-time statistical analysis capabilities

The traditional approach to assessing the statistical performance of O JIT-SDP algorithms involves using random seeds to construct classifiers on data streams [4,6,25,7]. However, this method encounters difficulties when applied to non-randomized algorithms and is not well-suited for parallel processing. More importantly, the dynamic nature of data instances means that online classifiers can change over time, which poses a challenge for capturing the classifier's evolving performance. The statistical results obtained from executing algorithms on a data stream at specific points in time do not fully represent the performance trajectory of the classifiers.

To overcome these challenges, we introduce a performance evaluation framework that integrates k-fold distributed validation with statistical testing, an approach originally proposed by Bifet et al. [14]. This framework enables real-time prequential evaluation and allows for pairwise comparisons between different classification algorithms. It also provides developers with the means to conduct continuous statistical testing throughout the entire prequential process, thereby enabling them to make informed, real-time decisions backed by robust statistical evidence. While this framework has been utilized in important fields such as recommender system [27], it has not been previously applied to the domain of online JIT-SDP. In contrast, many studies in this field have continued to employ the conventional evaluation methods, which, as discussed, have significant limitations.

Fig. 3 illustrates the framework for evaluating performance. We use a distributed validation strategy to partition the input commits into k-folds. Subsequently, we subject each alternative algorithm to testing, evaluation, and training within the k-folds partitions. This approach generates k classifiers for each online classification algorithm concurrently, based on the k-fold partitions of the data stream. This process facilitates the generation of paired indicator values, akin to the representation in the table depicted in Fig. 3. Finally, statistical tests can be executed using the generated table within one run on the data stream.

The core concept of this process involves distributing data instances among k distinct versions (classifiers) of an algorithm. Specifically, we adopt k-fold distributed bootstrap-validation for statistical analysis. Upon the arrival of each commit, it is used for training in each classifier according to a weight from a Poisson(1) distribution, and used for testing by all the others. K-fold distributed bootstrap-validation involves training with approximately 63.2% of commits and testing with around 36.8% of commits. The training instances'

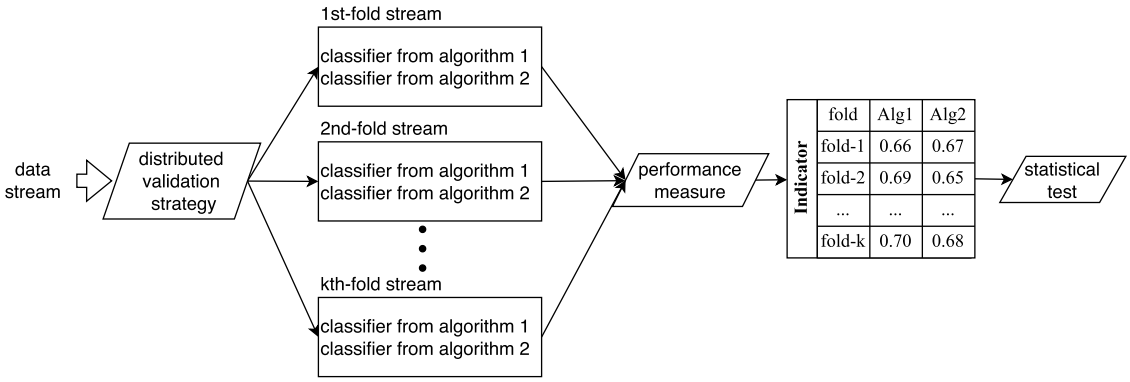


Fig. 3. The framework for online statistical performance evaluation.

Table 1
Allocation of 20 data points to 5 streams using k-fold distributed bootstrap validation.

Data Point	Stream A	Stream B	Stream C	Stream D	Stream E
1	T	S	T	T	S
2	T	T	S	T	T
3	S	T	T	S	T
4	T	S	T	T	T
5	T	T	S	S	T
6	S	T	T	T	S
7	T	S	T	T	T
8	T	T	S	T	S
9	S	T	T	S	T
10	T	S	T	T	T
11	T	T	S	T	S
12	S	T	T	T	T
13	T	S	T	S	T
14	T	T	S	T	S
15	S	T	T	T	T
16	T	S	T	T	T
17	T	T	S	T	S
18	S	T	T	S	T
19	T	S	T	T	T
20	T	T	S	T	S

overlap ratio between any two classifiers is approximately 40%, as computed based on $(1 - P(x = 0, \lambda = 1))^2$.To illustrate this allocation process, Table 1 presents a simulation where 20 data instances are distributed across five streams (A, B, C, D, E) using the k-fold distributed bootstrap validation strategy. In this simulation, each data instance is assigned as either Training (T) or Testing (S) instance for each classifier based on a Poisson(1) distribution. For example, when Data Point 1 enters the data stream, it is assigned as Training (T) instance in Streams A, C, and D, and as Testing (S) instance in Streams B and E according to the k-fold distributed bootstrap validation strategy (in simulation).

The online statistical performance evaluation framework presented above fulfills three essential requirements: independence from algorithms' randomness, parallelization, and real-time capability. First, it alleviates the need for evaluated classifiers to incorporate randomness, a critical aspect for online algorithms such as HoeffdingTree [26], HoeffdingAdaptiveTree [28], and HoeffdingOptionTree [29], which lack inherent randomness. Second, this approach partitions input data streams into parallel k-folds, generating statistical test results in a single execution. This efficiency is particularly beneficial for handling large datasets or evaluating complex classifiers, eliminating the necessity for repetitive executions to gather multiple observations. Third, the framework facilitates the computation of statistical performance metrics at any given moment, ensuring that the results accurately reflect the dynamic changes in the classifiers' performance over time. Despite the clear advantages of K-fold distributed validation outlined above, prior to our research, there has been no documented use of this approach in online defect prediction.

4. Study design

This section describes the experimental setup used to investigate existing challenges for HITL O-JIT-SDP to address in real-world applications.

4.1. Research questions

RQ1 (Validity of observed labels): To what extent can the observed performance derived from observed labels represent the true performance based on ground truth labels? We compare the observed performance derived from observed labels with the true performance based on ground truth labels. Through RQ1, we aim to quantify the discrepancy and understand the potential biases introduced by using observed labels in real applications of HITL O-JIT-SDP.

RQ2 (Impact of unrealistic normalization): What impact does using only the currently visible data for normalization have on the performance of the trained model? We compare the performance of HumLa before and after fixing unrealistic normalization in its code implementation. Through RQ2, we aim to understand the impact of data leakage in feature normalization on the performance of HITL O-JIT-SDP.

RQ3: (Setting of waiting time): How should the manual inspection time be realistically set in practice and what waiting time combinations achieve relatively optimal results? Through RQ3, we aim to identify suitable values for W_{QA} and W_{BFC} that will be utilized in the subsequent experiments within our study.

Afterward, we delve into pinpointing suitable statistical tests that enable immediate, on-demand comparisons of alternative classification algorithms amidst a continuous flow of commit data. This aspect is essential yet uncharted within the realm of O-JIT-SDP, lacking substantial evidence in existing literature.

RQ4: (Suitable statistical testing): What efficient and statistically valid methods should be employed in the context of O-JIT-SDP? Addressing RQ4 will lead us to recommend the optimal statistical test for conducting these comparisons effectively. The candidate statistical tests include McNemar's test, the Wilcoxon signed-rank test, and the Sign test (refer to Section 4.2 for details).

Finally, we culminate in a thorough comparison of prediction performance and assessment validity between HITL O-JIT-SDP and non-HITL O-JIT-SDP. The aim is to ascertain whether the incorporation of HITL into the prediction process amplifies prediction accuracy.

RQ5 (Benefits of HITL Integration): What benefits arise from the integration of HITL in the context of O-JIT-SDP? To address RQ5, we undertake a comparison between HITL O-JIT-SDP and non-HITL O-JIT-SDP in two aspects: (1) RQ5.1: their evaluation validity concerning the ideal O-JIT-SDP assessment approach, and (2) RQ5.2: the predictive performances of an identical online classification algorithm implemented within each framework. Through the exploration of RQ5, our goal is to comprehensively unravel the positive implications associated with the inclusion of HITL within the JIT defect prediction process.

4.2. Datasets

For RQ1 and RQ2, we utilize the dataset provided by the replication kit of HumLa, as these inquiries aim to examine the impact of HumLa's setups on its performance evaluation. RQ3 to RQ5 introduce W_{QA} and W_{BFC} setups, distinct from those of HumLa. Consequently, obtaining observed labels necessitates leveraging W_{QA}, W_{BFC}, and commit timestamps. However, the replication kit of HumLa does not furnish commit timestamps; instead, it offers processed datasets with observed labels under specific W_{BFC} settings.

Therefore, to address the remaining RQs, except for RQ1 and RQ2, we compile our datasets rather than directly employing those from HumLa. Precisely, we gather datasets with timestamp information from Commit Guru. It is worth noting that the projects in our datasets may not perfectly match those analyzed in HumLa. This discrepancy arises from ongoing updates to projects on Commit Guru, rendering some projects inaccessible during our dataset collection period. Additionally, certain projects on Commit Guru may have undergone increased commits since their inclusion in HumLa. Projects in our dataset were chosen randomly from a pool of projects that met the following criteria: a lifespan exceeding two years, containing over a total of 9,000 commits, and remaining active until at least 2021. The selection process ensures that the chosen projects are representative and suitable for addressing the research questions comprehensively.

Table 2 presents ten GitHub open-source projects utilized in our experiment. The brief descriptions of the ten projects used in the dataset are as follows: brackets[1] is a web development code editor with modern features and a sleek interface; cppcheck[2] is a static analysis tool for C/C++ code, ensuring code quality and identifying potential issues; edx-platform[3] is an online course learning platform designed for delivering courses at scale; FFmpeg[4] is a powerful multimedia framework for decoding, encoding, and streaming audio and video; gerrit[5] is a web-based code review and project management tool for Git-based repositories; gimp[6] is a versatile graphics editor with advanced features for image manipulation and creation; git[7] is a distributed version control system for tracking changes in source code during software development; mindspore[8] is a new open source deep learning training/inference

[1] https://github.com/adobe/brackets.
[2] https://github.com/danmar/cppcheck.
[3] https://github.com/openedx/edx-platform.
[4] https://github.com/FFmpeg/FFmpeg.
[5] https://github.com/GerritCodeReview/gerrit.
[6] https://github.com/GNOME/gimp.
[7] https://github.com/git/git.
[8] https://github.com/mindspore-ai/mindspore.

Table 2
Dataset used in our study.

Project	Total changes	%defect-inducing changes	Time period start	Time period end	Defects/ day	Commits/ day	Bug-fixing/ day
brackets	11914	29.4%	2011/12/7	2019/3/5	1.3	4.5	1.2
cppcheck	23033	36.4%	2007/5/7	2021/1/31	1.7	4.6	1.8
edx-platform	36932	26.6%	2011/12/7	2020/11/30	3.0	11.3	2.7
FFmpeg	91806	22.6%	2000/1/1	2020/12/7	2.7	12.0	3.7
gerrit	11794	28.0%	2008/10/21	2019/1/18	0.9	3.2	1.1
gimp	46586	32.2%	1997/1/1	2021/1/6	1.7	5.3	1.4
git	43772	28.7%	2005/10/10	2020/11/29	2.3	7.9	2.6
mindspore	6887	40.6%	2020/3/23	2020/11/11	12.0	29.6	9.7
pip	8495	24.8%	2008/10/15	2021/1/20	0.5	1.9	0.4
vlc	86936	29.6%	1999/8/8	2020/9/27	3.3	11.3	3.5
average	36815.5	29.9%	-	-	2.9	9.1	2.8

framework that could be used for mobile, edge and cloud scenarios; pip[9] is the package installer for Python, allowing users to easily install and manage software packages; and vlc[10] is a multimedia player and framework that supports various audio and video formats.

The data acquisition for these ten projects was facilitated by Commit Guru, an analytical tool adept at extracting commit-level datasets and annotating BFCs. Commit Guru is capable of extracting 14 distinct metrics spanning five categories: diffusion, size, purpose, history, and experience. For a comprehensive understanding of these metrics, please refer to the original research conducted by Kamei et al. [2]. In order to obtain the ground truth labels for commits in our datasets, we established a link between the BFCs identified by Commit Guru and their corresponding bug-inducing commits. To ensure the accuracy of these "ground truth" labels, we allowed each commit a significant amount of time to potentially have its associated BFC identified. We excluded commits from the final period of the data stream (specifically, the last two years) from our datasets. This is because these commits are relatively recent, and the BFCs linked to them may not have surfaced yet or we cannot confidently determine their ground truth labels. As a result, the statistics presented in Table 2 are calculated based on data streams ranging from the initial period to the period excluding the last two years. The complete dataset can be accessed in our open-source replication kits [11].

4.3. Performance indicators

To evaluate our HITL O-JIT-SDP framework and its corresponding online distributed k-fold evaluation framework, we employed three performance indicators: R_1, FPR (equal to 1 - R_0), and G-mean. R_0 signifies the recall value for clean commits, R_1 corresponds to the recall pertaining to commits that introduce bugs, and G-mean represents the geometric mean of R_0 and R_1. It is worth noting that most, if not all, of the existing O-JIT-SDP studies, chose R_0, R_1, and G-mean as their performance indicators for performance evaluation [4–7]. Although there is not a significant distinction between reporting FPR and reporting R_0, we specifically report FPR in our study to underscore the significance of achieving a low FPR for the effective practical implementation of O-JIT-SDP. Meanwhile, to conduct a comprehensive experimental evaluation, we also report the results of RQ5 in terms of F_1, MCC, and Precision in Appendix D [11]. The definitions of these three metrics are as follows:

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall},$$

Where

$$Precision = \frac{TP}{TP + FP},$$

$$Recall = \frac{TP}{TP + FN}$$

And

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Here, TP, TN, FP, and FN represent the number of true positive, true negative, false positive, and false negative commits, respectively. A higher Recall, Precision, F_1, G-mean and MCC value indicates better model performance.

Furthermore, since HumLa uses G-mean and MCC to evaluate classifiers, we adopt G-mean and MCC as performance indicators in RQ1 and RQ2 to maintain consistency with the original HumLa evaluation. In Appendix C [11], we report the results of RQ1 and RQ2 in terms of F_1 and Precision to conduct comprehensive evaluation.

[9] https://github.com/pypa/pip.
[10] https://github.com/videolan/vlc.

Table 3
Assumptions and applicability of statistical tests for comparing O-JIT-SDP classifiers.

Statistical Test	Assumptions	Verification with Dataset	Remarks
Wilcoxon Signed-Rank Test	Data is paired and comes from the same distribution.	Classifiers are compared on the same k data stream.	Satisfied.
	Differences between pairs are symmetrically distributed around the median.	The differences between pairs are likely to be approximately symmetric around the median because the random k-fold splits represent the overall data stream distribution [14].	Reasonably satisfied.
	Data is ordinal or continuous.	Performance metrics such as F_1 are continuous data.	Satisfied.
Sign Test	Data is paired.	Same paired data as used for the Wilcoxon Signed-Rank Test, satisfying this assumption.	Satisfied.
	Data is ordinal or continuous.	Performance metrics are continuous data.	Satisfied.
McNe- mar's Test	Data is paired and categorical (e.g., binary outcomes).	Predictions of classification models (e.g., buggy vs. clean) are categorical.	Satisfied.
	Outcomes are dichotomous (e.g., correct vs. incorrect).	Binary classification labels (e.g., buggy vs. clean) ensure dichotomous outcomes.	Satisfied.
	Test focuses on discordant pairs (instances where models differ in predictions).	Comparisons are conducted among different classifiers and should get discordant pairs in classification predictions under adequate sample size.	Satisfied.

The performance indicators mentioned earlier are computed using a fading factor α of 0.99 within a 10-fold distributed bootstrap-validation framework employed in our implementations of HITL O-JIT-SDP, except in RQ1 we follow the original setting of HumLa which uses a fading factor α of 0.99 on a single datastream rather than k-fold datastreams.

4.4. Statistical tests over commit stream

In the realm of online learning research [14,30], three non-parametric statistical tests stand out: the Wilcoxon signed-rank test, the sign test, and McNemar's test. These tests serve to determine whether one classification algorithm holds a statistically significant advantage over another classification algorithm. Remarkably, all three tests are capable of providing real-time test results through k-fold distributed validation. For these three statistical tests, we have summarized in Table 3 (1) their assumptions and (2) whether experiments comparing different O-JIT-SDP classifiers on the same dataset using k-fold distributed validation meet these assumptions. It is evident that the assumptions for all three tests are satisfied when comparing two classifiers using k-fold distributed validation. Consequently, further experimentation is needed to identify the most suitable test for this scenario, focusing on minimizing Type I and Type II error rates. Thus, we identify them as prime candidates for suitable statistical tests in the context of O-JIT-SDP.

McNemar's test. The test statistic for McNemar's test, which is a variant of the chi-squared test, involves counting two variables for a pair of classifiers. These variables are denoted as follows: A represents the number of instances that are misclassified by Algorithm 1 but correctly classified by algorithm 2, while B represents the number of instances that are misclassified by algorithm 2 but correctly classified by Algorithm 1.

$$\chi^2 = (A - B)^2 / (A + B) \tag{1}$$

The test statistic in McNemar's test follows a chi-squared distribution with one degree of freedom. To determine the statistical significance, the p-value (denoted as p) is compared to a predetermined significance level α, typically set at 0.05. If $p < \alpha$, we reject the null hypothesis, indicating a statistically significant difference between the two algorithms. Otherwise, given the current evidence, we cannot reject the null hypothesis, i.e., we have no evidence to show that there is a significant difference in performance between the two algorithms.

Wilcoxon signed-rank test. The Wilcoxon signed-rank test, unlike McNemar's test, requires calculation across multiple folds of performances (referred to as trials) for the algorithms being compared. Each fold represents a trial, and the performance on each fold is considered as an observation. To perform the test, the following test statistic using is calculated:

$$T = min(W+, W-) \tag{2}$$

where $W+$ is the sum of the ranks of the positive differences (where algorithm 2 outperforms Algorithm 1) and $W-$ is the sum of the ranks of the negative differences (where Algorithm 1 outperforms algorithm 2). For the test statistic T, the corresponding p-value (denoted as p) can be obtained using the normal distribution.

Sign test. The sign test, similar to the Wilcoxon signed-rank test, is performed on multiple folds of performances for the algorithms being compared. However, unlike the Wilcoxon signed-rank test, the sign test focuses on the direction of the differences between paired observations, rather than their magnitude. It involves comparing the signs of the differences between the paired observations. First, calculate the differences between paired observations. Assign a plus sign (+) to observations where the difference is positive, a minus sign (-) to observations where the difference is negative, and exclude observations where the difference is zero. Then, count the

number of plus signs and the number of minus signs. Finally, use the binomial distribution to calculate the p-value, which represents the probability of observing the given distribution of plus and minus signs by chance. The sign test is robust against outliers and does not rely on specific assumptions about the underlying distribution of the data, making it a useful non-parametric test in various scenarios.

To the best of our knowledge, prior research has not specifically utilized the mentioned non-parametric statistical tests (including McNemar's test, Wilcoxon signed-rank test, and the sign test) within the framework of k-fold distributed validation to assess JIT-SDP under the prequential setting. Although Wilcoxon Signed-rank tests have been utilized in previous O-JIT-SDP investigations (albeit without employing k-fold distributed validation), the substantiation for their use has been insufficient. In this study, we conduct an extensive experimental analysis within the domain of O-JIT-SDP, presenting statistical evidence to bolster their application. This contribution provides crucial empirical backing, filling a notable void in the existing research landscape. In Section 4.5.3, we outline an experiment aimed at determining the suitability of these tests for comparing classification algorithms within a runtime environment. Through this experiment, we aim to gain valuable insights into their performance and efficacy when applied to JIT-SDP classifiers, thereby aiding in the establishment of a dependable and appropriate evaluation methodology for such systems.

4.5. Data analysis methodology

4.5.1. Set up for RQ1 and RQ2

To rigorously explore RQ1 and RQ2, we first replicated the original execution of HumLa as presented in [10] to establish a performance baseline. Subsequently, we made targeted modifications to the HumLa source code to facilitate comparative analysis. The detailed results of these experiments, along with the specific modifications made to the source code, are extensively discussed in [13].

The experimental settings in RQ1 and RQ2 follow the original settings in HumLa. Specifically, experiments in RQ1 and RQ2 are conducted on the same 14 projects as HumLa, and each online learning process is repeated 100 times. The waiting time for BFC is set to 15 days. We use the default human noise setting of HumLa, which assumes zero false inspection from SQA staff. Oversampling-based Data Streaming bagging with confidence (ODaSC) [31] is used as the online classification algorithm. The implementation of HumLa (including its online learning process and its online classification algorithm) is based on an online learning framework skmultiflow [32]. HumLa uses the first 500 commits in a data stream to pre-train its online classifier and the rest of the commits in a data stream to incrementally train and test its online classifier. G-mean and MCC (F1 and Precision in Appendix C) with a fading factor of 0.99 are used to evaluate classifiers. Specifically, the average performance across each test commit in the whole data stream is calculated.

In both RQ1 and RQ2, we used the Wilcoxon signed-rank test to compare the performance differences of the online classifiers. The Benjamini-Hochberg procedure [33] was applied to correct the p-values. The motivation behind using this correction is to control for the family-wise error rate, as the multiple comparisons in our experiments could lead to an inflated type I error rate. Besides, Cliff's delta effect size is utilized to quantify the disparity between performances ("L" for large, "M" for medium, "S" for small, and "N" for negligible).

4.5.2. Set up for RQ3

In RQ3, We employ evaluation validity to gauge the degree to which a HITL O-JIT-SDP assessment accurately mirrors the prediction performance of the classifier under an "ideal" O-JIT-SDP evaluation scenario. Fig. 4(a) illustrates the ideal O-JIT-SDP evaluation process. In this ideal scenario, ground truth labels become instantly available as commits emerge, with no delay in label acquisition (waiting time set to 0). For a classifier undergoing the ideal O-JIT-SDP evaluation process, we denote its performance at timestamp ts as E_{ts}^{ideal}.

As described in Fig. 2, observed labels in HITL O-JIT-SDP are determined by W_{QA} and W_{BFC}. We denote the classifier's performance at timestamp ts under the HITL O-JIT-SDP evaluation process as $E_{ts}^{HITL}(W_{QA}, W_{BFC})$. In their work [6], Song et al. define the evaluation validity of HITL O-JIT-SDP as:

$$V_{ts} = 1 - |E_{ts}^{ideal} - E_{ts}^{HITL}(W_{QA}, W_{BFC})| \tag{3}$$

A larger value for V_{ts} indicates a better validity of the HITL O-JIT-SDP performance evaluation procedure. In RQ3, we explore the impact of SQA waiting time W_{QA} on evaluation validity by varying its value among 1, 3, 7, 15, 60, 90 days, while keeping W_{BFC} fixed at 15 days (as 15 days is the default BFC waiting time in [6]). Similarly, we investigate the impact of BFC waiting time W_{BFC} on evaluation validity by varying its value among 1, 3, 7, 15, 60, 90 days, while maintaining W_{QA} fixed at seven days.

In this paper, evaluation validity is computed using G-mean, which is derived from the average value of 10-fold bootstrap validation. The default online classification algorithm used to build classifiers is HoeffdingTree, the default algorithm provided by MOA [12]. It is noteworthy that we diverge from the approach of Song et al. [10], who employed ODaSC [31] as the online classification algorithm. To further explain the differences between our classifiers and those used in HumLa, we present Table 4, which compares the key features of the classifiers used in our evaluation with those employed in HumLa. This deviation stems from a practical consideration: ODaSC [31] was implemented on the scikit-multiflow online learning platform, which lacks support for distributed k-fold online validation. To incorporate real-time statistical evaluation into our HITL O-JIT-SDP, we opt for MOA, which facilitates distributed k-fold online validation, thus enabling the construction of our framework. Furthermore, given that ODaSC is a complex algorithm not yet implemented under MOA, we opt for online classification algorithms supported by MOA instead of re-implementing ODaSC on MOA to prevent implementation bias.

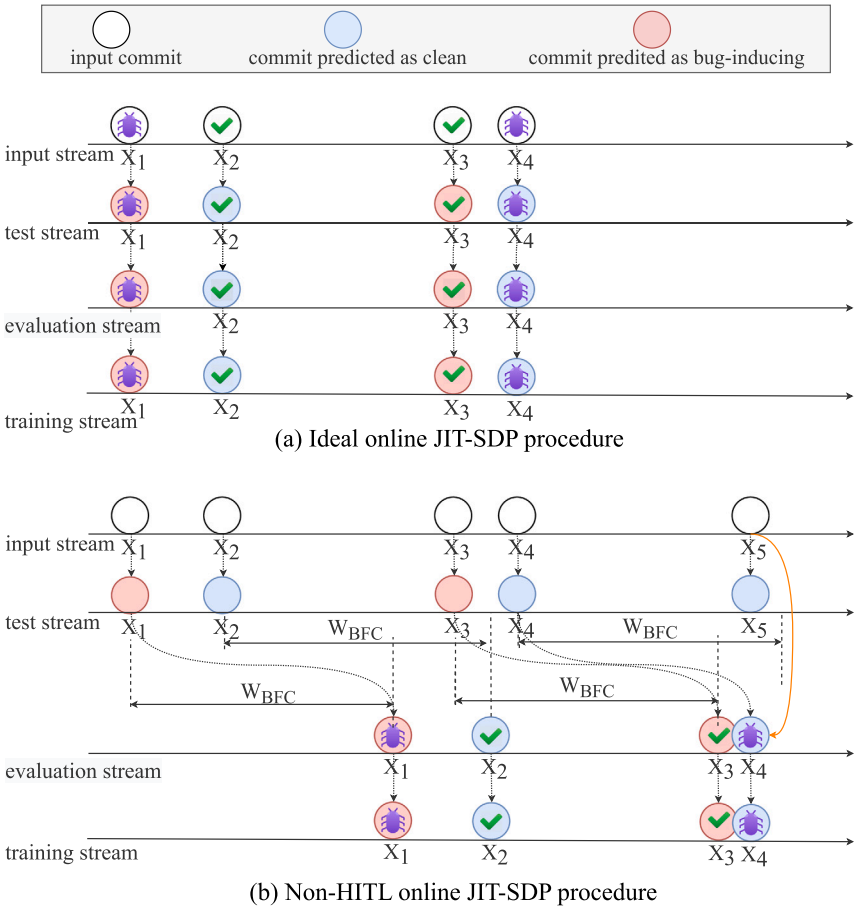


Fig. 4. Data flow in (a) ideal O-JIT-SDP and (b) non-HITL O-JIT-SDP.

Table 4
Comparison between classifiers used in our evaluation and in HumLa.

Feature	Our Classifier	HumLa Classifier
Algorithm description	A decision tree algorithm designed for streaming data, enabling efficient and incremental learning in online environments	An online classification algorithm that leverages bagging techniques combined with oversampling and confidence measures
Implementation Platform	MOA	scikit-multiflow
Platform Compatibility	Supports distributed k-fold online validation	Lacks distributed k-fold validation support
Implementation Complexity	Utilizes existing MOA algorithms to prevent implementation bias	Requires re-implementation of ODaSC on MOA, introducing potential bias
Statistical Evaluation	Incorporates real-time statistical evaluation within HITL O-JIT-SDP	Limited by platform constraints for real-time evaluation

4.5.3. Set up for RQ4

In this study, the comparison of classification algorithms entails evaluating classifiers (as instances of algorithms) on identical data streams. When considering the suitability of statistical tests for comparing O-JIT-SDP classifiers in runtime within the framework of k-fold distributed bootstrap-validation, we examine the following three tests: McNemar's test, the Wilcoxon signed-rank test, and the Sign test. The chosen statistical test should demonstrate sensitivity to both Type I and Type II errors while comparing O-JIT-SDP classifiers in runtime. Type I error occurs when a true null hypothesis is erroneously rejected. In our scenario, this error arises when the statistical test incorrectly identifies two classifiers with similar performances as having different performances. On the other hand, Type II error arises when a false null hypothesis is not rejected. In our context, this error manifests when the statistical test incorrectly concludes that two classifiers with different performances exhibit similar performances. It is imperative to underscore that selecting an appropriate statistical test demands robust empirical evidence regarding its Type I and Type II error characteristics.

To detect Type I errors in statistical tests, we utilize a method involving two classifiers intentionally constructed to perform equally, which are then executed on identical data streams. Each classifier is generated using randomized seeds for the same algorithm.

We employ k-fold distributed bootstrap validation to generate multiple k-fold streams and develop two sets of k-fold classifiers. Subsequently, we subject paired observations from these classifiers to statistical tests. If a statistical test rejects the null hypothesis, suggesting a significant difference in performance between the two algorithms when there should not be one, it results in a Type I error.

To identify Type II errors in statistical tests, we utilize a method where two classifiers, deliberately designed to perform equally, are executed on identical data streams. Each classifier is constructed using the same algorithm but with different noise filters. For instance, at a noise level of 0.05, we introduce a filter that randomly flips the predicted classification (from 0 to 1 or from 1 to 0) with a probability of 0.05. We employ k-fold distributed bootstrap-validation to generate multiple k-fold streams and develop two sets of k-fold classifiers on them. Subsequently, statistical tests are applied to paired sets of k observations. If a statistical test fails to reject the null hypothesis, indicating that the performance difference between two classifiers built from the same algorithm with different noise filters is not statistically significant, then the test commits a Type II error.

The above experimental methodology follows the original design by Bifet et al. [14] and the base algorithm used to build classifiers in this experiment is same with [14]: LeverageBagging [34], a HoeffdingTree-based online classification with randomness, to make the experiment work. In RQ4, we consider classifier noise levels of 0, 0.05, 0.1. The chosen confidence level for our statistical tests is 0.05. Our evaluation framework allows for real-time calculation of these statistical tests. However, to optimize computational resources, we have implemented a strategy that avoids performing statistical tests on every single observation. Instead, we adopt a sampling approach. At regular intervals, specifically every N/10th example in the stream, where N represents the total number of commits in the stream, we sample observations. This strategy ensures that each stream is sampled a total of ten times, thereby generating ten sets of k-folds observations. Subsequently, each statistical test (including the Wilcoxon signed-rank test and signed test) is carried out on these paired sets of 10-folds observations, resulting in a single p-value per test. It is important to highlight that McNemar's test involves a slightly distinct approach. Instead of being computed on k-fold observations as the other tests, each McNemar's test is carried out on instances within a single fold. This process is then repeated k times, yielding a comprehensive perspective on its statistical significance.

In the context of each data stream, if the p-value resulting from the comparison between Classifiers1 (Algorithm with $randomseed_1$) and Classifier2 (Algorithm with $randomseed_2$) is less than 0.05, it signifies the occurrence of a Type I error. Conversely, when the p-value stemming from the comparison between Classifier1 (Algorithm with noise = 0 and fixed $randomseed$) and Classifier2 (Algorithm with noise = 0.05 and fixed $randomseed$) surpasses 0.05, it indicates the commission of a Type II error. Similarly, if the p-value derived from the comparison between Classifier1 (Algorithm with noise = 0 and fixed $randomseed$) and Classifier2 (Algorithm with noise = 0.1 and fixed $randomseed$) surpasses 0.05, it corresponds to a Type II error. By implementing the aforementioned methodology, we iteratively apply each type of statistical test to each project, conducting this process 50 times to enable the calculation of the Type I error rate and Type II error rate for every statistical test.

4.5.4. Set up for RQ5

In RQ5, we undertake comparisons between HITL O-JIT-SDP and non-HITL O-JIT-SDP in two aspects: evaluation validity comparison in RQ5.1 and predictive performances comparison in RQ5.2. Fig. 4(b) depicts non-HITL O-JIT-SDP evaluation process. In this scenario, all observed labels are determined by a waiting time window W_{BFC} and BFCs. For a classifier operating in non-HITL O-JIT-SDP, we denote its performance at timestamp ts as $E_{ts}^{non-HITL}(W_{BFC})$. Therefore, the evaluation validity of non-HITL O-JIT-SDP is similar with the evaluation validity of HITL O-JIT-SDP in Section 4.5.2:

$$V_{ts} = 1 - |E_{ts}^{ideal} - E_{ts}^{non-HITL}(W_{BFC})| \tag{4}$$

In RQ5.1, we establish equivalent waiting times for both HITL and non-HITL O-JIT-SDP scenarios, and subsequently compare their evaluation validity. This involves pitting HITL with SQA labeling (waiting time: $W_{QA} = 15$ days, $W_{BFC} = 15$ days) against non-HITL with BFC labeling (waiting time: $W_{BFC} = 15$ days). The objective here is to investigate whether SQA labeling performed by HITL can enhance the evaluation validity of O-JIT-SDP, even when the time duration matches that of BFC labeling. Furthermore, we introduce an additional aspect by considering a shorter waiting time for SQA labeling (HITL with $W_{QA} = 7$ days, $W_{BFC} = 15$ days) against non-HITL with BFC labeling ($W_{BFC} = 15$ days). This additional comparison aims to determine whether a swifter SQA labeling process can further enhance the evaluation validity of HITL O-JIT-SDP.

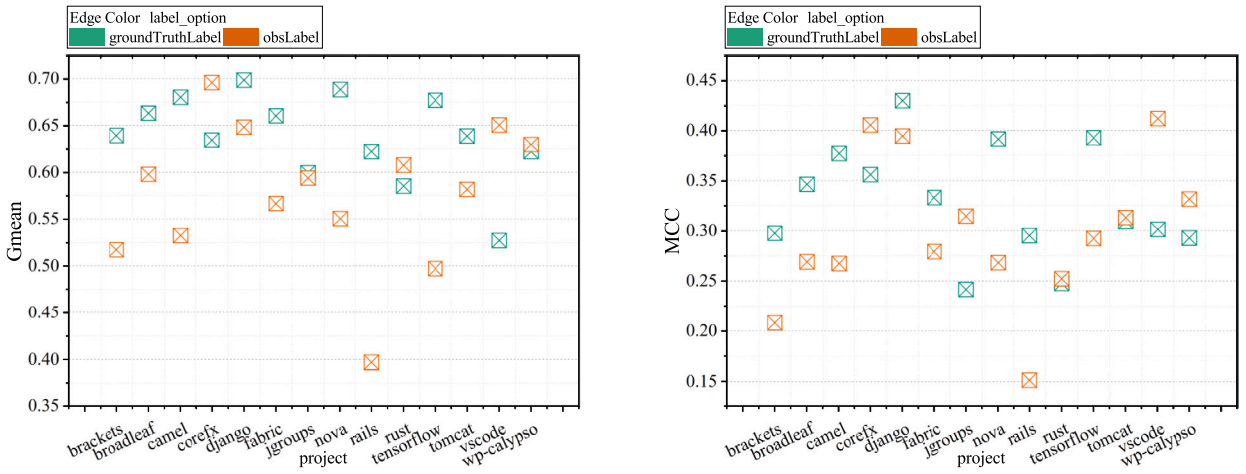
In RQ5.2, our focus shifts to the comparative analysis of prediction performance between HITL and non-HITL O-JIT-SDP scenarios. We conduct real-time monitoring of both G-mean performance and the specific statistical test highlighted in RQ3. This monitoring encompasses HITL(W_{QA}, W_{BFC}) and non-HITL(W_{BFC}) contexts, enabling us to discern variations in prediction performance between these two approaches.

5. Experimental results

In this section, we provide a thorough account of the experimental outcomes.

5.1. RQ1: Validity of observed labels

To address this question, we assess the predictive accuracy of HITL O-JIT-SDP models by comparing their performance using ground truth labels versus observed labels.



(a) G-mean (b) MCC

Fig. 5. The average performance of HumLa before and after replacing the usage of ground truth labels with observed labels.

Table 5
The G-mean performance of HumLa when evaluated with ground truth labels and observed labels.

Project	Using GT labels	Using observed labels	perf bias	p-value
brackets	0.639(0.003)	0.517(0.005)	-19.08%	0.000[L]
broadleaf	0.663(0.003)	0.598(0.007)	-9.83%	0.000[L]
camel	0.680(0.004)	0.532(0.012)	-21.75%	0.000[L]
fabric	0.660(0.004)	0.567(0.011)	-14.19%	0.000[L]
jgroups	0.600(0.009)	0.594(0.022)	-0.95%	0.000[N]
nova	0.688(0.001)	0.550(0.003)	-20.05%	0.000[L]
tomcat	0.639(0.004)	0.582(0.004)	-8.91%	0.129[L]
corefx	0.634(0.011)	0.696(0.009)	9.69%	0.000[L]
django	0.698(0.002)	0.648(0.005)	-7.21%	0.000[L]
rails	0.622(0.004)	0.397(0.013)	-36.15%	0.000[L]
rust	0.585(0.015)	0.607(0.017)	3.80%	0.000[L]
tensorflow	0.677(0.003)	0.497(0.005)	-26.57%	0.000[L]
vscode	0.527(0.009)	0.650(0.010)	23.43%	0.000[L]
wp-calypso	0.622(0.003)	0.630(0.006)	1.20%	0.000[L]
Average	0.638(0.005)	0.576(0.009)	-9.04%	-

Fig. 5 illustrates the average performance across 100 runs of HumLa on each project. The green square represents HumLa's performance when evaluated with ground truth labels, while the orange square represents its performance when evaluated with observed labels. As depicted in Fig. 5, substituting ground truth labels with observed labels during HumLa evaluation led to a decrease in observed performance across ten out of 14 projects in terms of G-mean (a decrease of 71.4% of the projects) and across eight out of 14 projects in terms of MCC (a decrease of 57.1% of the projects). This indicates that in most datasets, practical application of HITL results in observed performance inferior to the model's performance under ideal evaluation conditions.

Table 5 and Table 6 present the average performance metrics and their standard variations for HumLa across each project. The final row depicts the mean values across all projects. In the p-value columns, red text highlights p-values smaller than 0.05, indicating non-negligible effect sizes. The "perf bias" columns denote the performance bias of observed labels, calculated as:

$$\frac{Ind(Observed\,HumLa\,performance) - Ind(ideal\,HumLa\,performance)}{Ind(ideal\,HumLa\,performance)} \tag{5}$$

From Table 5 and Table 6, we find a significant distinction between ideal and observed HumLa performance, as most p-values related to G-mean and MCC fall below 0.05. However, the average performance bias of observed HumLa stands at -9.4% for G-mean and -9.1% for MCC, which might not be considered substantial.

Answer for RQ1: *In real-world applications, performance evaluation relies on observed labels rather than ground truth labels. In 71.4% of cases, the observed G-mean is inferior to that evaluated using ground truth labels, while in 57.1% of cases, the observed MCC falls short of its ground truth counterpart. Furthermore, performance assessed with observed labels consistently proves significantly inferior to that derived from ground truth labels.*

Table 6
The MCC performance of HumLa when evaluated with ground truth labels and observed labels.

Project	Using GT labels	Using observed labels	perf bias	p-value
brackets	0.297(0.004)	0.208(0.004)	-30.09%	0.000[L]
broadleaf	0.346(0.004)	0.269(0.007)	-22.29%	0.000[L]
camel	0.377(0.008)	0.267(0.008)	-29.03%	0.000[L]
fabric	0.333(0.008)	0.279(0.012)	-16.13%	0.000[L]
jgroups	0.242(0.013)	0.314(0.027)	30.16%	0.000[L]
nova	0.391(0.003)	0.268(0.003)	-31.48%	0.000[L]
tomcat	0.309(0.004)	0.313(0.005)	1.25%	0.000[L]
corefx	0.356(0.011)	0.405(0.017)	13.87%	0.000[L]
django	0.430(0.003)	0.394(0.005)	-8.21%	0.000[L]
rails	0.295(0.004)	0.151(0.007)	-48.68%	0.050[L]
rust	0.248(0.024)	0.252(0.030)	1.84%	0.000[N]
tensorflow	0.393(0.005)	0.292(0.005)	-25.59%	0.000[L]
vscode	0.301(0.009)	0.412(0.013)	36.71%	0.000[L]
wp-calypso	0.293(0.003)	0.332(0.006)	13.31%	0.000[L]
Average	0.329(0.007)	0.297(0.011)	-8.17%	-

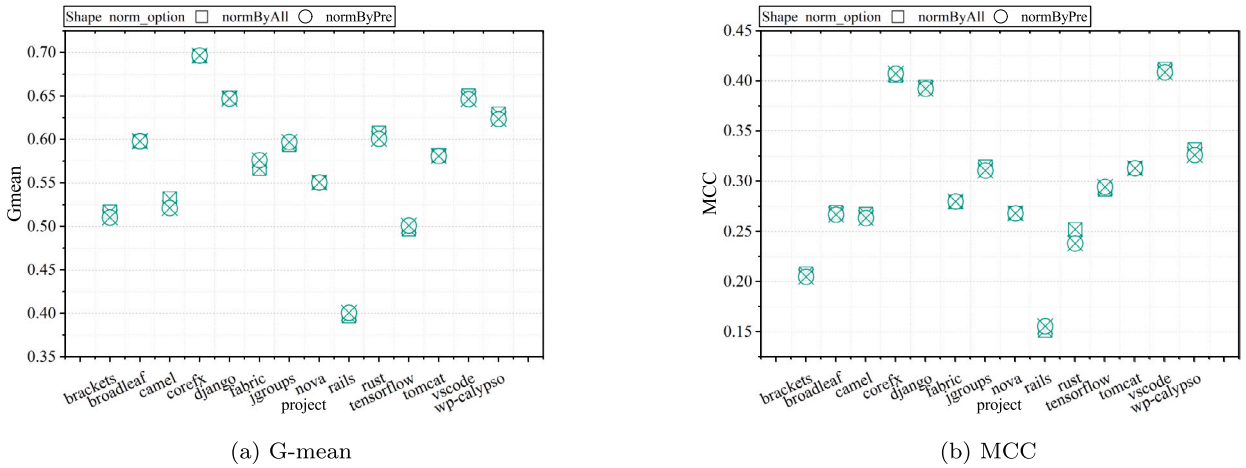


Fig. 6. The average observed performance of HumLa using all data for normalization and using historical data for normalization.

5.2. RQ2: Impact of unrealistic normalization

In Fig. 6, it is evident that substituting the data leakage normalization strategy with normalizing commits solely based on pre-training commits yields a negligible change in observed performance concerning G-mean and MCC.

In Table 7 and Table 8, we delve deeper into the statistical contrast between normalizing all data and normalizing historical data. Specifically, Table 7 and Table 8 present the average observed performance values and their standard variation of HumLa across each project, alongside the Benjamini-Hochberg corrected p-value for comparing HumLa performances when utilizing all data for normalization versus historical data. The last row provides the average value across all projects. In the p-value columns, red text indicates a p-value smaller than 0.05, signaling a non-negligible effect size. The term "perf bias" is defined as:

$$\frac{Ind(NormByHistorical) - Ind(normByAll)}{Ind(normByAll)} \tag{6}$$

In 57.1% of cases, the G-mean decreases after employing realistic normalization, while in 64.3% of cases, the MCC decreases. Additionally, considering the p-values, we observe values smaller than 0.05 for eight out of 14 projects concerning both G-mean and MCC. This indicates that the contrast between unrealistic normalization and its correction holds statistical significance across the majority of projects.

Answer for RQ2: *In 57.1% of cases, the G-mean decreases after applying realistic normalization, and in 64.3% of cases, the MCC decreases. Although the overall difference remains small in most cases, the performance change is statistically significant in 6 out of 14 projects. Therefore, while addressing unrealistic normalization is important, its impact on HITL O-JIT-SDP performance is not always essential.*

Table 7
Observed G-mean performance of HumLa using all data for normalization and using historical data for normalization.

Project	Normalized by all	Normalized by historical	perf bias	p-value
brackets	0.517(0.005)	0.510(0.008)	-1.32%	0.000[L]
broadleaf	0.598(0.007)	0.598(0.006)	-0.02%	0.853[N]
camel	0.532(0.012)	0.521(0.014)	-2.06%	0.000[L]
fabric	0.567(0.011)	0.576(0.008)	1.71%	0.519[L]
jgroups	0.594(0.022)	0.597(0.020)	0.54%	0.149[N]
nova	0.550(0.003)	0.551(0.003)	0.04%	0.000[N]
tomcat	0.582(0.004)	0.581(0.004)	-0.17%	0.398[S]
corefx	0.696(0.009)	0.697(0.009)	0.11%	0.974[N]
django	0.648(0.005)	0.647(0.005)	-0.18%	0.100[S]
rails	0.397(0.013)	0.401(0.016)	0.90%	0.011[S]
rust	0.607(0.017)	0.601(0.018)	-1.10%	0.000[S]
tensorflow	0.497(0.005)	0.501(0.007)	0.82%	0.069[M]
vscode	0.650(0.010)	0.646(0.007)	-0.62%	0.003[S]
wp-calypso	0.630(0.006)	0.623(0.011)	-0.97%	0.000[M]
Average	0.576(0.009)	0.575(0.010)	-0.17%	-

Table 8
Observed MCC performance of HumLa using all data for normalization and using historical data for normalization.

Project	Normalized by all	Normalized by historical	perf bias	p-value
brackets	0.208(0.004)	0.205(0.006)	-1.49%	0.001[M]
broadleaf	0.269(0.007)	0.267(0.007)	-0.82%	0.069[S]
camel	0.267(0.008)	0.260(0.010)	-1.48%	0.001[S]
fabric	0.279(0.012)	0.280(0.010)	0.22%	0.422[N]
jgroups	0.314(0.027)	0.311(0.027)	-1.17%	0.011[N]
nova	0.268(0.003)	0.268(0.003)	0.04%	0.555[N]
tomcat	0.313(0.005)	0.313(0.005)	-0.01%	0.419[N]
corefx	0.405(0.017)	0.407(0.017)	0.48%	0.797[N]
django	0.394(0.005)	0.392(0.005)	-0.54%	0.000[S]
rails	0.151(0.007)	0.156(0.008)	2.73%	0.002[M]
rust	0.252(0.030)	0.238(0.032)	-5.59%	0.014[S]
tensorflow	0.292(0.005)	0.294(0.006)	0.78%	0.950[S]
vscode	0.412(0.013)	0.409(0.011)	-0.78%	0.084[S]
wp-calypso	0.332(0.006)	0.326(0.008)	-1.65%	0.000[M]
Average	0.297(0.011)	0.295(0.011)	-0.66%	-

5.3. RQ3: Setting of waiting time

Two factors potentially influence the evaluation validity of HITL O-JIT-SDP. Fig. 7 reports the influence of W_{QA} on evaluation validity with a fixed $W_{BFC} = 15$ days, while Fig. 8 reports the influence of W_{BFC} with a fixed $W_{QA} = 7$ days. Both figures are calculated based on G-mean at the 1000^{th} commit, chosen because it represents a stable point after the cold-start phase but before model degradation [35] becomes significant, ensuring a reliable assessment of classifier performance. Additionally, the use of a fading factor ensures that the performance at this point reflects an average over a recent window of commits. Moreover, we include the experimental results of RQ3 at the 2000^{th}, 3000^{th}, 4000^{th}, and 5000^{th} commits in Appendix A [11] and the conclusions obtained at these timestamps are consistent with those at the 1000^{th} commit.

From Fig. 7, we can make the following observations: Firstly, the evaluation validity of online learners differs significantly when W_{QA} equals zero compared to when W_{QA} is non-zero. This implies that if we assess online learners using $W_{QA} = 0$, their actual performance in real-world scenarios (where W_{QA} is not zero) may not be accurately represented.

Secondly, the evaluation validity tends to be higher when the SQA waiting time is less than 15 days for the majority of projects. Generally, a shorter SQA waiting time correlates with higher evaluation validity. However, since SQA feedback depends on the time SQA staff spend reviewing suspect commits, we recommend a 7-day SQA waiting time as a reasonable trade-off. This waiting time balances the need for timely feedback with providing SQA personnel sufficient time to perform thorough, high-quality assessments.

Analyzing Fig. 8, we observe two key points: First, evaluation validity varies significantly across different BFC waiting time windows. For example, the “brackets” project fluctuates between 57% and 96%, while the “vlc(master)” project ranges from 56% to 98%. This highlights the importance of selecting an appropriate BFC waiting time window for optimal evaluation validity. Second, a 15-day window proves to be suitable for many projects, aligning with prior research on O-JIT-SDP [6], which has established 15 days as the default BFC waiting time in our analysis.

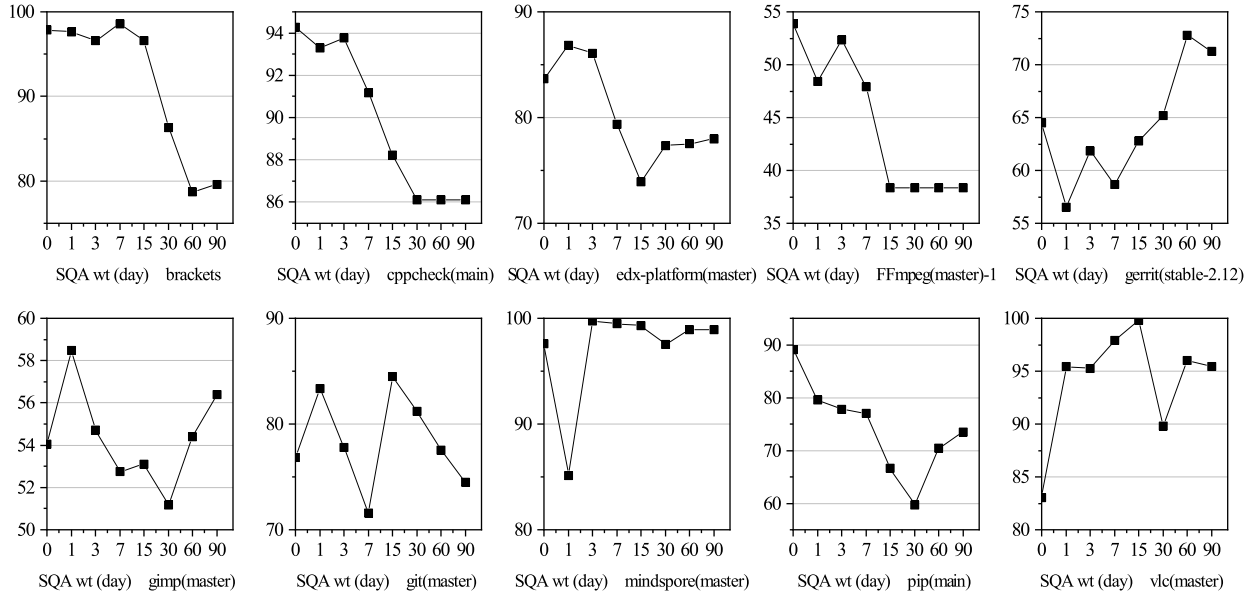


Fig. 7. Impact of SQA waiting time (x-axis) on evaluation validity of HITL O-JIT-SDP (y-axis), with a fixed W_{BFC} of 15 days.

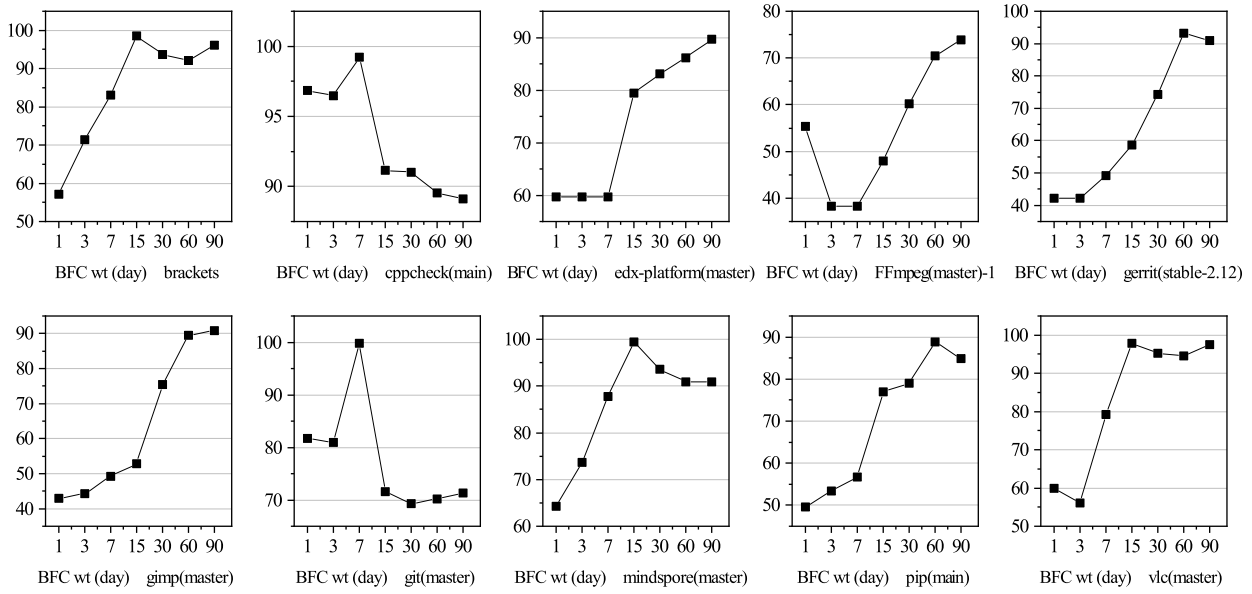


Fig. 8. Impact of BFC waiting time (x-axis) to evaluation validity of HITL O-JIT-SDP (y-axis), with a fixed W_{QA} of 7 days.

Answer for RQ3: *The waiting time windows for SQA and BFC significantly influence the evaluation validity in HITL O-JIT-SDP. In our experiment, we've established default waiting time windows of 7 days for SQA and 15 days for BFC, reflecting the typical validity observed across various projects.*

5.4. RQ4: Suitable statistical testing

Table 9 summarizes the outcomes of statistical tests. It displays the percentages of null hypotheses rejected in McNemar's test, Wilcoxon signed-rank test, and sign test across 10-fold bootstrap validation. The tests were carried out to compare algorithms under three conditions: unchanged (same algorithm with varied initialization seed), changed with a noise level of 0.05, and changed with a noise level of 0.1.

From Table 9, we have the following observations:

Table 9
The ratio of null hypothesis rejection when using McNemar's test, Wilcoxon signed-rank test, and sign test under 10-fold bootstrap-validation.

	Mcnemar			Wilcoxon			Sign test		
Project	No Change	Change Noise =0.05	Change Noise =0.1	No Change	Change Noise =0.05	Change Noise =0.1	No Change	Change Noise =0.05	Change Noise =0.1
brackets	0.205	0.850	0.880	0.080	1.000	1.000	0.040	0.960	1.000
cppcheck	0.100	0.885	0.900	0.160	1.000	1.000	0.080	1.000	1.000
edx-platform	0.205	0.900	0.900	0.020	0.740	1.000	0.020	0.720	1.000
FFmpeg	0.060	0.900	0.900	0.060	0.920	1.000	0.020	0.560	0.760
gerrit	0.110	0.870	0.900	0.120	1.000	1.000	0.100	1.000	1.000
gimp	0.145	0.900	0.900	0.020	1.000	1.000	0.020	1.000	1.000
git	0.115	0.900	0.905	0.000	1.000	1.000	0.000	0.980	1.000
mindspore	0.075	0.805	0.860	0.040	1.000	1.000	0.060	1.000	1.000
pip	0.010	0.875	0.895	0.140	0.580	0.900	0.100	0.540	0.800
vlc	0.150	0.900	0.900	0.000	1.000	1.000	0.020	0.940	1.000
Average	0.118	0.879	0.894	0.064	0.924	0.990	0.046	0.870	0.956

- In terms of the Type I error ratio, McNemar's test exhibits a misleading characteristic by falsely identifying non-existent differences around 11.8% of the time. Consequently, it demonstrates a pronounced inclination to overstate the disparities between algorithms. Conversely, both the Wilcoxon signed rank test and the sign test display comparable Type I error rates, at 6.4% and 4.6% respectively.
- In terms of Type II error ratio, the Wilcoxon signed rank test excels in distinguishing between algorithms. On average across all projects, it achieves null hypothesis rejection rates of 92.4% for change noise = 0.05 and 99.0% for change noise = 0.1. The sign test's average rejection rates are 87.0% for change noise = 0.05 and 95.6% for change noise = 0.1, while McNemar's test achieves averages of 87.9% for change noise = 0.05 and 89.4% for change noise = 0.1.

Answer for RQ4: *We recommend employing the Wilcoxon signed-rank test to assess the statistical significance of performance differences among the O-JIT-SDP algorithms within a k-fold distributed bootstrap-validation framework. McNemar's test is not advisable due to its elevated Type I error rate when comparing O-JIT-SDP algorithms.*

5.5. RQ5: Benefits of HITL Integration

5.5.1. Does the incorporation of HITL result in improved evaluation validity?

To address this question, we begin by comparing the evaluation validity of HITL and non-HITL O-JIT-SDP under identical long waiting times. The evaluation validity depicted in Fig. 9 is derived from the G-mean calculated on the 1000^{th} commit. Due to space constraints, the experimental results at the 2000^{th}, 3000^{th}, 4000^{th}, and 5000^{th} commits can be found in online Appendix B [11]. The conclusions in the appendix are consistent with those in this section. Meanwhile, in Fig. 10, the evaluation validity is computed based on the G-mean across the first 1000^{th}, 2000^{th}, 3000^{th}, 4000^{th}, and 5000^{th} commits.

In Fig. 9, it becomes evident that, in the majority of instances, HITL O-JIT-SDP attains superior evaluation validity in comparison to its non-HITL counterpart. Notably, when examining scenarios such as FFmpeg(master)-1, gimp(master), git(master), pip(main), and vlc(master), the HITL evaluation approach consistently achieves heightened evaluation validity across varying waiting durations. Similarly, for other projects, the HITL evaluation process either secures higher or comparable evaluation validity across nearly all waiting periods. As outlined in section 3.1, the quality of labels can be elevated by leveraging SQA feedback to facilitate the aggregation of labeled commits. Fig. 9 reveals that the integration of SQA feedback for collecting labeled commits indeed augments evaluation validity when juxtaposed with solely relying on BFC waiting time for commit collection, even if the duration of SQA waiting time mirrors that of BFC.

Furthermore, the HITL O-JIT-SDP evaluation process offers the advantage of providing SQA feedback more rapidly than the standard BFC waiting period, which averages 15 days. As illustrated in Table 2, the daily averages for total commits and bug-fixing commits are 9.1 and 2.8, respectively. With these figures in mind, we postulate that SQA staff could potentially review a flagged commit within a week. This assumption prompts us to explore the potential enhancement of evaluation validity with a reduced SQA waiting period, especially when compared to the non-HITL O-JIT-SDP approach. Fig. 10 illustrates that, across most projects and observation points (#commit), HITL($W_{QA} = 7$, $W_{BFC} = 15$) achieves superior evaluation validity compared to HITL($W_{QA} = 15$, $W_{BFC} = 15$) and non-HITL($W_{BFC} = 15$) scenarios.

Answer for RQ5.1: *HITL enhances O-JIT-SDP not only when SQA feedback outpaces BFC feedback (by offering training commits with superior label quality in a shorter duration) but also when the waiting time for SQA feedback equals that of BFC feedback (as it consistently delivers training commits with enhanced label quality).*

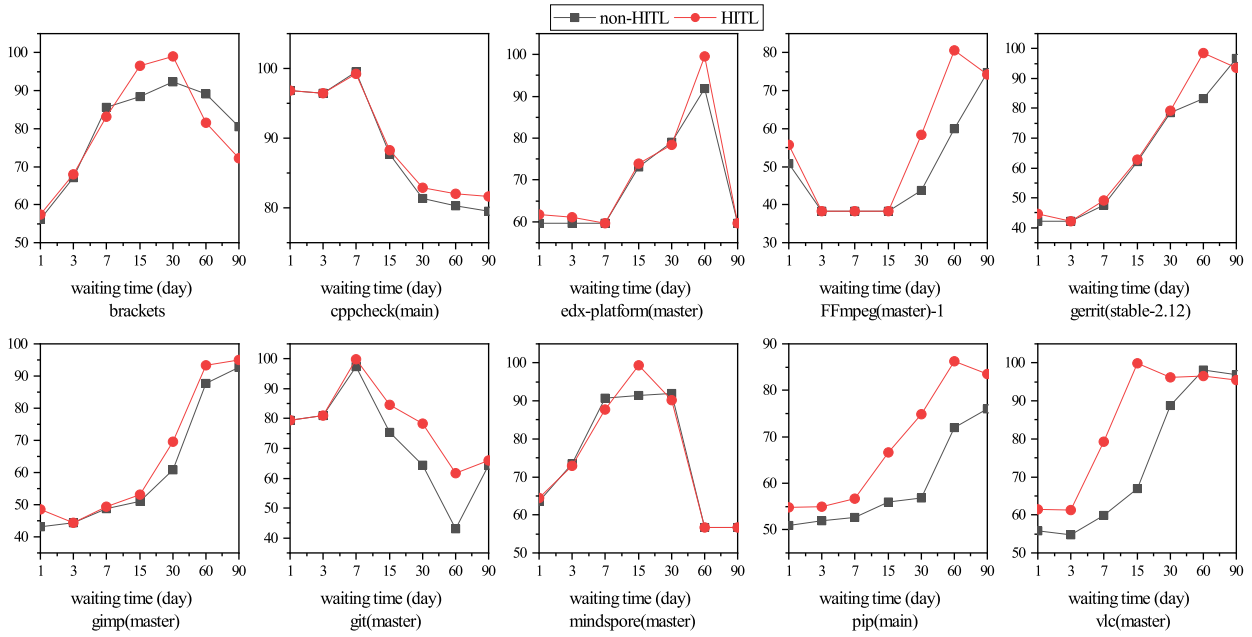


Fig. 9. Comparison of the evaluation validity (y-axis) between HITL(W_{QA} = W, W_{BFC} = W) vs. non-HITL(W_{BFC} = W). The x-axis is the value of W, which varies among 1,3,7,15,30,60,90 days.

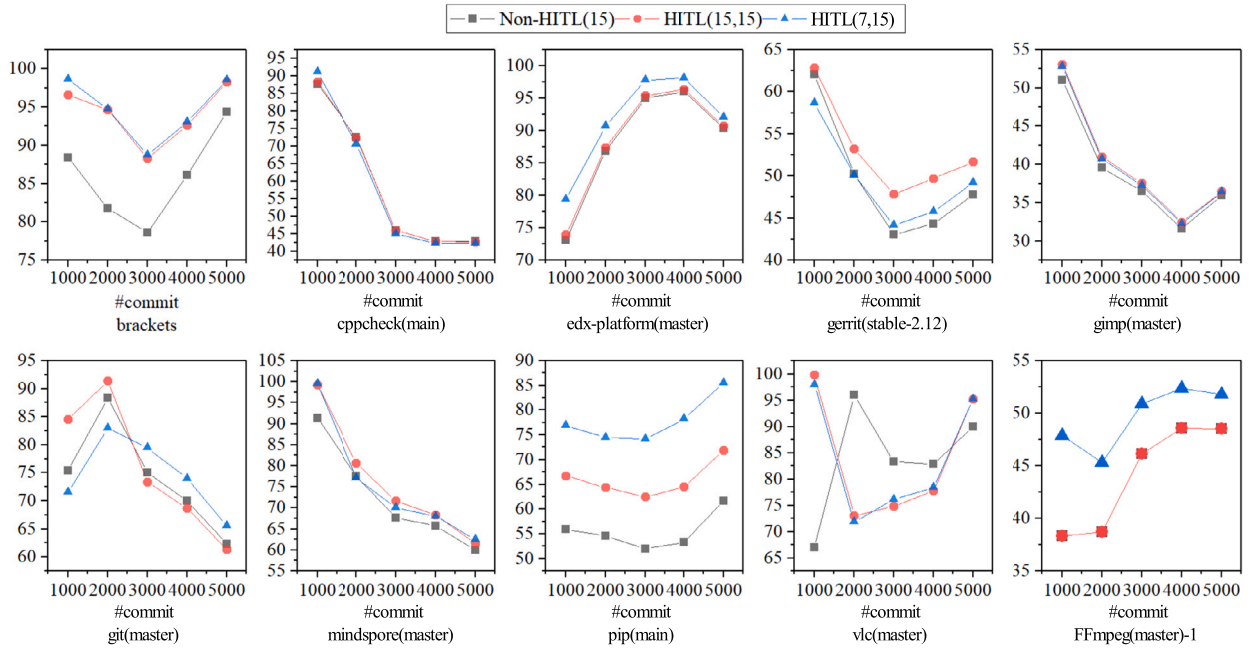


Fig. 10. Comparison of the evaluation validity (y-axis) between HITL(W_{QA}, W_{BFC}) vs. non-HITL(W_{BFC}). The x-axis is the number of top commits under evaluation.

5.5.2. Does the incorporation of HITL result in improved prediction performance?

In this section, we conduct a comparative analysis of the G-mean performance between HITL and non-HITL O-JIT-SDP. Due to space limitations, we utilize two projects as illustrative examples to illustrate the real-time evolution of G-mean and the outcomes of Wilcoxon signed-rank tests applied to G-mean. As depicted in Fig. 11, the performance of the same classifier, HoeffdingTree, exhibits notable superiority in terms of G-mean when operating under the HITL paradigm as compared to the non-HITL approach, particularly following a brief period of unstable cold start. The results of the Wilcoxon signed-rank test affirm this observation, indicating that the classifier within the HITL O-JIT-SDP framework consistently outperforms its non-HITL counterpart after the initial cold start

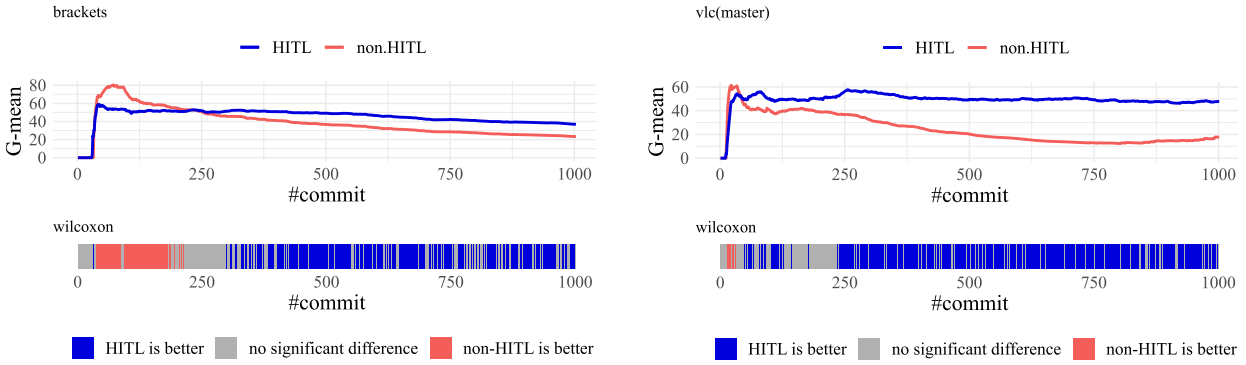


Fig. 11. Evolution of G-mean under HITL vs. non-HITL and the corresponding real-time Wilcoxon signed-rank test result before the first 1000 commits. HITL denotes HITL($W_{QA} = 7$, $W_{BFC} = 15$) and non-HITL denotes non-HITL($W_{BFC} = 15$).

phase, across the majority of instances. Note that figures for all ten projects can be accessed within our replication kit [11], and they consistently demonstrate a parallel trend across the entirety of projects.

Answer for RQ5.2: *Under HITL, classification algorithms generally outperform non-HITL O-JIT-SDP counterparts.*

6. Discussion

In this section, we begin by examining the influence of human noise on the performance evaluation of HITL O-JIT-SDP. Following that, we explore the potential enhancement of HITL O-JIT-SDP performance through resampling technology. We then summarize the essential insights for both researchers and practitioners. Finally, we discuss potential threats that could affect the validity of our study.

6.1. Impact of human noise on observed classifier performance

In the study by Song et al. in HumLa [10], the effects of human noise on the performance of online classifiers were examined. Our study builds upon this by proposing an evaluation framework that utilizes the observed performance of these classifiers. We next delve into the influence of human noise on the observed performance metrics, focusing on the scenarios that arise when a commit predicted as positive by the online classifier receives an incorrect label from SQA staff. Specifically, two outcomes are possible: (1) a true positive (tp) commit is mislabeled as negative, and (2) a false positive (fp) commit is mislabeled as positive. These scenarios lead to the creation of a set of all tp commits, denoted as TP, and a set of all fp commits, denoted as FP, which are used to calculate performance metrics. Under the HITL O-JIT-SDP framework, all commits reviewed by SQA staff are initially predicted as positive, which means that scenario (1) transforms a tp into an fp, thereby reducing the count of $|TP|$ and increasing the count of $|FP|$. In contrast, scenario (2) transforms an fp into a tp, increasing the count of $|TP|$ and decreasing the count of $|FP|$. An essential question arises: which of these scenarios will predominantly alter the observed performance? Assuming humans err with equal likelihood for each commit, the frequency of these scenarios hinges on the proportion of $|TP|$ and $|FP|$ relative to $|TP + FP|$. If the online classifier generates more fp commits than tp commits, human inspection will more frequently result in scenario (1) over scenario (2). With human noise denoted by x, and provided $|FP|$ exceed $|TP|$, the observed number of true positives ($|TP|'$) will increase with human noise ($|TP|' = |TP| - -|TP| \times x + |FP| \times x$), while the observed number of false positives ($|FP|'$) will decrease ($|FP|' = |FP| - |FP| * x + |TP| * x$). Consequently, the online classifier, being incrementally updated with more instances bearing positive observed labels (as $|TP|'$ increases), will tend to predict more instances as positive. This leads to an increase in both the observed Recall and the False Positive Rate (FPR).

Regarding the HT classifier utilized in our experiments, Fig. 12 illustrates its performance ranking under varying levels of human noise within the HITL O-JIT-SDP framework. As human noise increases, the observed Recall generally improves (better rank), while the observed FPR typically worsens (poorer rank), suggesting the HT classifier predicts a higher number of commits as positive. Initially, the observed G-mean increases with human noise (when noise is less than 0.5), then it generally declines (when noise exceeds 0.5). This suggests that for the HT classifier on our datasets, there is a tendency for a higher incidence of FPs compared to TPs.

In conclusion, the challenge of detecting TPs in O-JIT-SDP tasks, where positive instances are exceedingly rare, is a prevalent issue. In these situations, some classifiers with limited predictive capabilities may exhibit a higher frequency of FPs over TPs. As human noise escalates, an increasing number of FPs are misclassified as TPs, prompting the online classifier to issue more positive predictions. Ultimately, this trend results in a tendency for the observed Recall and FPR of the online classifier to rise.

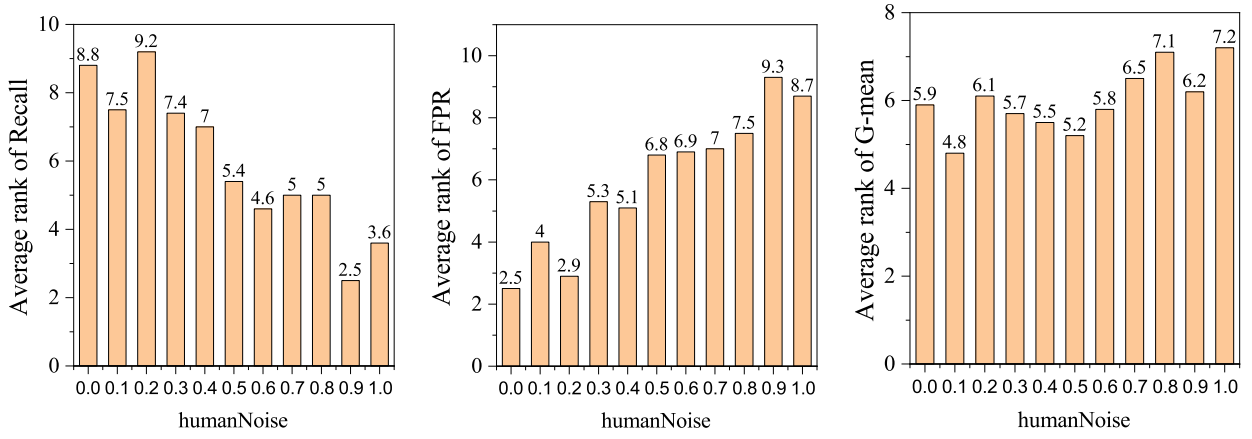


Fig. 12. Performance rank of HT at different human noise under HITL(W_{QA} = 7, W_{BFC} = 15). Column plots report average ranks of classifier performances across projects. Performance on 1000^{th} commit is reported. The smaller the rank, the better the performance.

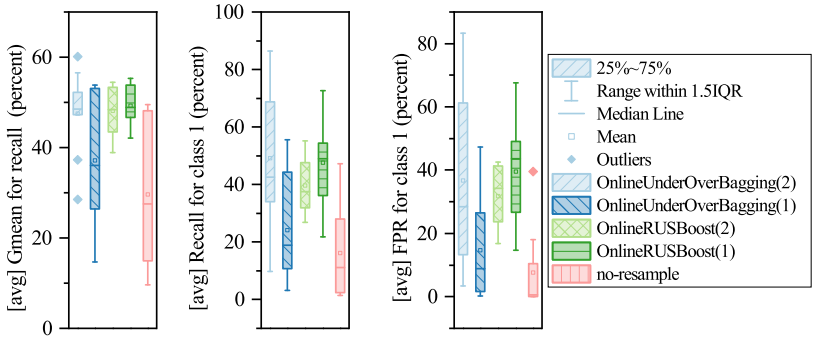


Fig. 13. Performance distribution across ten projects of Hoeffding Tree classifier with and without resampling strategies. Under the evaluation of HITL(7,15). Resampling rates of one or two adjust the positive/negative class ratio, with a rate of two representing a 200% positive-to-negative instances ratio.

6.2. Impact of data resampling

While prior studies have examined resampling strategies within the context of O-JIT-SDP [5], the improvement of model performance through resampling strategies remains untested within the framework of HITL O-JIT-SDP. In this study, we employ the OnlineUnderOverBagging [36] and OnlineRUSBoost [36] resampling strategies. The former involves under-sampling the majority class and over-sampling the minority class, while the latter integrates resampling techniques as a post-processing step before each iteration of the standard AdaBoost algorithm, aligning with the techniques of OnlineUnderOverBagging. We implement these strategies using the MOA toolkit [12]. We did not include the resampling strategies used in previous O-JIT-SDP research [5], since they were implemented under scikit-multiflow and not easy to be implemented under MOA. Fig. 13 illustrates the performance distributions of HoeffdingTree with and without the integration of resampling strategies. Our observations are as follows:

- *Enhanced R_1/G-mean.* It is evident that nearly all resampling strategies lead to a noticeable improvement in R_1 performance, regardless of the chosen resampling rate, except for OnlineUnderOverBagging(1). Notably, a significant enhancement in G-mean and R_1 is apparent when comparing classifiers employing resampling against the no-resample HoeffdingTree, as evident in Fig. 13.
- *Degraded FPR.* The majority of resampling configurations adversely affect FPR. The classifiers utilizing resampling consistently exhibit higher FPRs in comparison to the no-resample HoeffdingTree. Furthermore, a concerning trend is observed: approximately three-quarters of resampled classifiers have a mean FPR exceeding 30% (36.7% for OnlineUnderOverBagging(2), 31.8% for OnlineRUSBoost(2), and 39.6% for OnlineRUSBoost(1)), while the mean FPR of the no-resample scenario remains considerably lower at only 7.7%.

In summary, resampling strategies tend to enhance the performance of HITL O-JIT-SDP when aiming for improvements in G-mean or R_1. However, it is important to note that such an enhancement often comes at the cost of an increased FPR. In the practical context of defect prediction, a high FPR is undesirable, as it can lead to an excessive number of false alarms, which developers are unwilling to address [37]. Therefore, when selecting a resampling strategy for O-JIT-SDP models, careful consideration of the FPR's impact is essential.

6.3. Implications

Our research holds significant implications for both practitioners and researchers in the field.

6.3.1. For practitioners

SQA practitioners fulfill a dual role as both users of O-JIT-SDP systems and integral participants in the operational dynamics of these systems. Within the realm of O-JIT-SDP systems, the HITL variant emerges as a more pragmatic selection for SQA practitioners, primarily due to its heightened realism in contrast to the non-HITL version. Within the HITL framework, online algorithms demonstrate superior performance in comparison to their non-HITL counterparts. This trend can be attributed primarily to the increased dependability and efficiency of commit labels provided by SQA practitioners, outperforming the labels generated solely through the "waiting time window + BFC" approach. Another significant observation pertains to the adaptability of feedback timing that SQA practitioners necessitate. Our experiments underscore that a delay in feedback ranging from one to 15 days does not substantially compromise the validity of evaluations. This suggests that SQA practitioners can meticulously assess predicted bug-inducing commits at their own pace, thereby ensuring the delivery of feedback of the highest quality.

6.3.2. For researchers

Our study introduces a practical O-JIT-SDP model and an evaluation framework tailored for O-JIT-SDP. First, when proposing a new O-JIT-SDP algorithm, we advocate for its assessment within the HITL O-JIT-SDP environment to gauge its performance under realistic conditions. Second, while evaluating O-JIT-SDP algorithms, we recommend adopting k-fold distributed bootstrap-validation, which offers an appropriate ratio of training/testing instances and a reasonable overlap ratio of k-fold streams. Third, in the context of O-JIT-SDP, the Wilcoxon signed-rank test is a more fitting statistical tool compared to the sign test and McNemar's test, given its balanced consideration of Type I and Type II error rates. Fourth, while resampling strategies can enhance R_1 and G-mean metrics, they also elevate FPR. It is imperative to be mindful of FPR when evaluating preprocessing strategies or classification algorithms within the realm of O-JIT-SDP.

6.4. Threats to validity

Construct validity. The main threat is the potential disparity between our initial expectations regarding the role of SQA staff and their actual functions within the O-JIT-SDP framework. Uniquely, our research addresses the interaction dynamics between SQA staff and the O-JIT-SDP system, which we regard as a reflection of practical development practices. In our experimental setup, we make the assumption that SQA staff unfailingly possess accurate insights into the true labels of commits following an SQA inspection waiting period. However, this presumption might not align with reality, given that SQA staff can be prone to errors despite their considerable prior knowledge of the program. Nonetheless, our conviction remains steadfast that SQA involvement can yield swifter and more precise label feedback compared to the "waiting time window + BFC" approach. In this context, our assertion that "HITL improves O-JIT-SDP" retains its validity. Another important aspect that requires further consideration is the potential threat to validity arising from noise introduced by the ground truth label collection. In our experiments, the ground truth labels of commits are collected by waiting for BFC commits for at least two years, which means a commit is labeled as "clean" only if no corresponding BFC commit shows up for two years. Still, some commits may be wrongly labeled as "clean" and cause label noises. This issue stands out as one of the principal threats to the validity of O-JIT-SDP research including ours.

Internal validity. The main threat is the potential limitation posed by the extent of our investigation into SQA waiting time and BFC waiting time. Given the constraints of space, we were unable to execute our experiment across a broader spectrum of SQA and BFC waiting times for RQ1. Nonetheless, we have meticulously chosen representative waiting time values, encompassing 1, 3, 7, 15, 30, 60 days, and our replication kit stands poised to facilitate deeper exploration in the future. Furthermore, it is imperative to acknowledge that our experimental foundation relies on the HoeffdingTree, an established online classification algorithm. While our conclusions are not universally generalizable to all online algorithms, it is worth noting that HoeffdingTree enjoys status as a classic and widely utilized algorithm within the context of O-JIT-SDP [4–7]. Nevertheless, the avenue remains open for further experimentation encompassing a range of algorithms through the extension of the MOA framework [13].

External validity. The main threat to our study is the potential lack of generalizability of our findings to other projects. We have gathered data from ten distinct open-source projects using Commit Guru, a widely employed tool in prior O-JIT-SDP research. These projects span various domains. While we are confident that this external challenge falls within an acceptable scope, we recognize that our outcomes may not be universally applicable. Furthermore, we specifically acknowledge that the findings associated with RQ4 are strictly derived from the specific datasets and experimental settings utilized in this study, and we do not intend to imply that the results are universally generalizable. In other words, the conclusions drawn for RQ4 reflect phenomena observed under our current experimental conditions. To tackle above issue, we have undertaken measures to bolster the external validity of our study. By making our code and data openly accessible, we facilitate effortless replication of our research in the times ahead.

7. Evolutionary trajectory of O-JIT-SDP

In this section, we trace the developmental trajectory of O-JIT-SDP from its foundational stages (JIT-SDP with consideration of time impact) to the current state focusing on Human-in-the-Loop (HITL) methodologies. In Table 10, We highlight key research in the development of O-JIT-SDP. As can be seen, O-JIT-SDP has traversed several pivotal stages, including time-sensitive JIT-SDP

Table 10
The development trajectory of O-JIT-SDP.

Author(s) [Reference]	Year	Formulation	Summary of Primary Contributions
Tan [3]	2015	time sensitive JIT-SDP	Adapt O-JIT-SDP to address the problems of JIT-SDP.
McIntosh [38]	2018	JIT-SDP	JIT-SDP models lose effectiveness over time due to evolving code properties. Regular retraining with recent data and using larger data caches improve model performance and reliability.
Cabral [5]	2019	O-JIT-SDP	Class imbalance evolution significantly affects JIT-SDP performance. ORB, a new O-JIT-SDP approach, provides improved predictive performance.
Tabassum [7]	2020	O-JIT-SDP	Using cross-project data alongside within-project data in O-JIT-SDP improves predictive performance, especially in the project's initial phase and during performance drops.
Song [6]	2022	O-JIT-SDP	Provide a mathematical formulation for continuous evaluation procedure of O-JIT-SDP predictive performance.
Cabral [4]	2022	O-JIT-SDP	Analyze the impact of concept drift on O-JIT-SDP performance, and introduces PBSA, a method to handle concept drift and verification latency.
Song [25]	2023	O-JIT-SDP	Smaller waiting times increase label noise, impacting JIT-SDP evaluation validity slightly.
Cabral [9]	2023	O-JIT-SDP	Offline base learners are suggested for performance improvements.
Song [10]	2023	HITL O-JIT-SDP	Introduces the first human labeling methods HumLa for O-JIT-SDP to improve the prediction performance.
Ours	2024	HITL O-JIT-SDP	Reformulate of the initial HITL O-JIT-SDP to enable its application in a realistic and industrial setting and reveal the effectiveness of HITL O-JIT-SDP under a practical condition.

formulation [38], class imbalance evolution [5], the utilization of cross-project training data [7], continuous evaluation procedure formulation [6], concept drift [4], impact of waiting time on evaluation validity [25], utilization of offline base learners [9], currently arriving at the HITL phase [10]. Moving forward, to ensure the sustained and healthy growth of this field, we propose a review of the problem formulation of HITL O-JIT-SDP.

To ensure ongoing progress in the defect prediction domain, we advocate for a revisiting of current formulations in defect prediction frameworks, addressing deficiencies in those frameworks with robust methodologies and evidence-based solutions. To achieve this goal, source code should be scrutinized carefully, avoiding model evaluations that are only feasible under ideal conditions (where experimental setups may guarantee model evaluation but are impractical or unrealistic in the real world). Guided by this principle, we have refined the entire formulation of the first O-JIT-SDP model, HumLa, encompassing both model training and evaluation processes.

8. Conclusion and future work

In this study, we have reformulates the initial HITL O-JIT-SDP research to make it more applicable in practical scenarios. We propose a generalized HITL O-JIT-SDP framework and examine the pitfalls of the initial and only (to the best of our knowledge) HITL O-JIT-SDP approach, HumLa, based on empirical evidence. Our study highlights practical considerations often overlooked in ideal experimental conditions. Our work aims to bolster the credibility of model evaluations and provide a foundation for future research in HITL O-JIT-SDP. This work underscores the significance of refining HITL frameworks to enhance their industrial applicability and pave the way for further advancements in the field.

In the future, we intend to apply the enhanced framework to a wider range of projects to further explore the value of HITL in online JIT-SDP. We also aim to assess its versatility and efficacy across various software engineering contexts, including predicting false positives in static analysis, forecasting crash-prone faults, and estimating bug severity. The ultimate goal of this research direction is to broaden our understanding of HITL's role in software engineering and to identify its potential applications across different domains.

CRediT authorship contribution statement

Xutong Liu: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Yufei Zhou:** Writing – review & editing, Writing – original draft, Project administration, Formal analysis, Data curation. **Yutian Tang:** Writing – review & editing, Project administration, Formal analysis, Data curation. **Junyan Qian:** Writing – review & editing, Project administration, Formal analysis, Data curation. **Yuming Zhou:** Writing – review & editing, Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work is partially supported by National Natural Science Foundation of China (62172205, 62162004, U21A20474).

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.scico.2025.103296.

References

[1] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, N. Ubayashi, An empirical study of just-in-time defect prediction using cross-project models, in: Proceedings of the 11th Working Conference on Mining Software Repositories, Association for Computing Machinery, 2014, pp. 172–181.
[2] Y. Kamei, E. Shihab, B. Adams, A. Hassan, A. Mockus, A. Sinha, N. Ubayashi, A large-scale empirical study of just-in-time quality assurance, IEEE Trans. Softw. Eng. 39 (6) (2013) 757–773.
[3] M. Tan, L. Tan, S. Dara, C. Mayeux, Online defect prediction for imbalanced data, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 2, IEEE, 2015, pp. 99–108.
[4] G. Cabral, L. Minku, Towards reliable online just-in-time software defect prediction, IEEE Trans. Softw. Eng. 49 (3) (2023) 1342–1358.
[5] G. Cabral, L. Minku, E. Shihab, S. Mujahid, Class imbalance evolution and verification latency in just-in-time software defect prediction, in: 2019 IEEE/ACM 41st International Conference on Software Engineering, 2019, pp. 666–676.
[6] L. Song, L. Minku, A procedure to continuously evaluate predictive performance of just-in-time software defect prediction models during software development, IEEE Trans. Softw. Eng. 49 (2) (2023) 646–666.
[7] S. Tabassum, L. Minku, D. Feng, G. Cabral, L. Song, An investigation of cross-project learning in online just-in-time software defect prediction, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, IEEE, 2020, pp. 554–565.
[8] S. Tabassum, L. Minku, D. Feng, Cross-project online just-in-time software defect prediction, IEEE Trans. Softw. Eng. 49 (1) (2023) 268–287.
[9] G. Cabral, L. Minku, A.L. Oliveira, D. Pessoa, S. Tabassum, An investigation of online and offline learning models for online just-in-time software defect prediction, Empir. Softw. Eng. 28 (5) (2023) 121.
[10] L. Song, L.L. Minku, C. Teng, X. Yao, A practical human labeling method for online just-in-time software defect prediction, in: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, 2023, pp. 605–617.
[11] X. Liu, liu906/hitl-online-just-in-time-defect-prediction: v1.0, https://doi.org/10.5281/zenodo.14575065, Dec. 2024.
[12] B. Albert, H. Geoff, P. Bernhard, K. Philipp, K. Hardy, J. Timm, S. Thomas, Moa: massive online analysis, a framework for stream classification and clustering, in: Proceedings of the First Workshop on Applications of Pattern Analysis, vol. 11, PMLR, 2010, pp. 44–50.
[13] X. Liu, liu906/moa-extension: v1.0, https://doi.org/10.5281/zenodo.14575053, Dec. 2024.
[14] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, B. Pfahringer, Efficient online evaluation of big data stream classifiers, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2015, pp. 59–68.
[15] J.a. Gama, R. Sebastião, P.P. Rodrigues, Issues in evaluation of stream learning algorithms, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2009, pp. 329–338.
[16] K. Malialis, C. Panayiotou, M. Polycarpou, Online learning with adaptive rebalancing in nonstationary environments, IEEE Trans. Neural Netw. Learn. Syst. 32 (10) (2021) 4445–4459.
[17] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, L. He, A survey of human-in-the-loop for machine learning, Future Gener. Comput. Syst. 135 (2022) 364–381.
[18] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, J. Xiao, Lsun: construction of a large-scale image dataset using deep learning with humans in the loop, preprint, arXiv:1506.03365, 2015.
[19] T. Grønsund, M. Aanestad, Augmenting the algorithm: emerging human-in-the-loop work configurations, J. Strateg. Inf. Syst. 29 (2) (2020) 101614.
[20] C. Wah, S. Belongie, Attribute-based detection of unfamiliar classes with humans in the loop, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 779–786.
[21] J. Peterson, R. Battleday, T. Griffiths, O. Russakovsky, Human uncertainty makes classification more robust, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 9617–9626.
[22] H. Wang, S. Gong, X. Zhu, T. Xiang, Human-in-the-loop person re-identification, in: Computer Vision–ECCV 2016: 14th European Conference, Computer Vision – ECCV 2016, Springer International Publishing, 2016, pp. 405–422.
[23] C. Chai, L. Cao, G. Li, J. Li, Y. Luo, S. Madden, Human-in-the-loop outlier detection, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, 2020, pp. 19–33.
[24] I. Lage, A. Ross, S.J. Gershman, B. Kim, F. Doshi-Velez, Human-in-the-loop interpretability prior, in: Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018.
[25] L. Song, L. Minku, X. Yao, On the validity of retrospective predictive performance evaluation procedures in just-in-time software defect prediction, Empir. Softw. Eng. 28 (5) (2023) 124.
[26] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, 2000, pp. 71–80.
[27] J. Vinagre, A.M. Jorge, C. Rocha, J. Gama, Statistically robust evaluation of stream-based recommender systems, IEEE Trans. Knowl. Data Eng. 33 (7) (2021) 2971–2982.
[28] A. Bifet, R. Gavaldà, Adaptive learning from evolving data streams, in: Advances in Intelligent Data Analysis VIII: 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31-September 2, 2009. Proceedings 8, Advances in Intelligent Data Analysis VIII, Springer Berlin Heidelberg, 2009, pp. 249–260.
[29] B. Pfahringer, G. Holmes, R. Kirkby, New options for hoeffding trees, in: AI 2007: Advances in Artificial Intelligence: 20th Australian Joint Conference on Artificial Intelligence, AI 2007: Advances in Artificial Intelligence, Springer Berlin Heidelberg, 2007, pp. 90–99.
[30] N. Japkowicz, M. Shah, Evaluating Learning Algorithms: A Classification Perspective, Cambridge University Press, 2011.
[31] L. Song, S. Li, L. Minku, X. Yao, A novel data stream learning approach to tackle one-sided label noise from verification latency, in: 2022 International Joint Conference on Neural Networks, 2022, pp. 1–8.
[32] J. Montiel, J. Read, A. Bifet, T. Abdessalem, Scikit-multiflow: a multi-output streaming framework, J. Mach. Learn. Res. 19 (1) (2018) 2915–2914.
[33] Y. Benjamini, Y. Hochberg, Controlling the false discovery rate: a practical and powerful approach to multiple testing, J. R. Stat. Soc., Ser. B, Methodol. 57 (1) (1995) 289–300.
[34] A. Bifet, G. Holmes, B. Pfahringer, Leveraging bagging for evolving data streams, in: Machine Learning and Knowledge Discovery in Databases: European Conference, Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, 2015, pp. 135–150.

[35] S. Liu, S. Xue, J. Wu, C. Zhou, J. Yang, Z. Li, J. Cao, Online active learning for drifting data streams, IEEE Trans. Neural Netw. Learn. Syst. 34 (1) (2023) 186–200.
[36] B. Wang, J. Pineau, Online bagging and boosting for imbalanced data streams, IEEE Trans. Knowl. Data Eng. 28 (12) (2016) 3353–3366.
[37] B. Johnson, Y. Song, E. Murphy-Hill, R. Bowdidge, Why don't software developers use static analysis tools to find bugs?, in: 2013 35th International Conference on Software Engineering, 2013, pp. 672–681.
[38] S. McIntosh, Y. Kamei, Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction, in: Proceedings of the 40th International Conference on Software Engineering, Association for Computing Machinery, 2018, p. 560.