Empirical Software Engineering (2024) 29:158
https://doi.org/10.1007/s10664-024-10551-8

Online cross-project approach with project-level similarity for just-in-time software defect prediction

Cong Teng[1,2] · **Liyan Song**[3] · **Xin Yao**[4]

Accepted: 23 September 2024 / Published online: 1 October 2024

Abstract
The adoption of additional Other Project (OP) data has shown to be effective for online Just-In-Time Software Defect Prediction (JIT-SDP). However, state-of-the-art online Cross-Project (CP) methods, such as All-In-One (AIO) and Filtering, which operate at the data-level, encounter the difficulties in balancing diversity and validity of the selected OP data, which can negatively impact predictive performance. AIO may select unrelated OP data, resulting in a lack of validity, while Filtering tends to select OP data that closely resemble Target Project (TP) data, leading to a lack of diversity. To address this validity-vs-diversity challenge, a promising approach is to utilize an online project-level OP selection methodology. This approach selects instructive other projects that exhibit similarities to TP and can positively impact predictive performance, achieving better data validity compared to AIO and maintaining higher diversity compared to Filtering. To accomplish this, we propose a project-level Cross-Project method with Similarity (CroPS), which employs appropriate project-level similarity metrics to identify instructive other projects for model updating over time. CroPS applies a specified threshold to determine the selection of other projects at any given moment. Furthermore, we propose an ensemble-like framework called Multi-threshold CroPS (Multi-CroPS), which incorporates multiple threshold options for selecting other projects and poses the importance of defect-inducing changes. Experimental results based on 23 open-source projects validate the effectiveness of our project-level metrics for calculating similarities between projects. The results also demonstrate that CroPS significantly enhances the predictive performance while reducing computational costs compared to existing data-level CP approaches. Moreover, Multi-CroPS achieves significantly better performance than state-of-the-art CP approaches including our CroPS.

Keywords Just-in-time software defect prediction · Cross-project · Online learning · Project-level similarity · Verification latency

1 Introduction

Software defects refer to defects in software products that can result in a failure in meeting software requirements or specifications, necessitating subsequent repairs (Zubrow 2009).

Communicated by: Meiyappan Nagappan

Extended author information available on the last page of the article

Timely detection of software defects is crucial to prevent significant economic losses (Newman 2002) and mitigate potential risks to life, as exemplified by incidents such as plane crashes caused by software defects (Favarò et al. 2013). Just-In-Time Software Defect Prediction (JIT-SDP) is a specialized type of software defect prediction that focuses on making prediction on the code change level, specifically aiming to determine whether a software change will induce a defect at the time of its commit (i.e., just-in-time) (Kamei et al. 2013; Kim et al. 2008; Kim and Whitehead Jr 2006). JIT-SDP adopts Machine Learning (ML) techniques to automate the process and produce high-quality defect predictions. In this context, JIT-SDP is a typically binary classification problem, with the class labels being *defect-inducing* and *clean* (Kamei et al. 2013; Song et al. 2023b; Kim et al. 2008).

To pursue good defect prediction, a substantial amount of training samples is typically required to build and continuously update the JIT-SDP model. However, practical scenarios often present challenges due to the limited availability of training samples, particularly during the initial stages of software project development or when *concept drift*, denoting changes in the software development process within the company, takes place. In such scenarios, the insufficiency of training samples can result in unsatisfactory prediction results, as the JIT-SDP models may lack the necessary amount of data to effectively capture the underlying patterns and characteristics of defect-inducing changes. To address this issue, training samples from other project(s), referred to as Other Project (OP) data, can be employed to enlarge the training set of the Target Project (TP). The aim is to enhance the predictive performance of online JIT-SDP by leveraging data from other projects (Kamei et al. 2016; Tabassum et al. 2022). The specialized process of employing OP data for model training is commonly referred to as *CP JIT-SDP*, following prior research studies (Tabassum et al. 2020, 2022).

Previous research on CP JIT-SDP has primarily focused on the offline scenario, where classifiers are constructed based on a pre-existing training set and are not updated thereafter (Zhang et al. 2022; Kamei et al. 2016; Shehab et al. 2022; Cho et al. 2018). Such CP JIT-SDP models remain static and cannot incorporate new information as the software development process progresses. Consequently, these offline models may find difficulty in capturing the evolving nature of software project development, leading to a potential lose in performance (Tabassum et al. 2022). In practical settings, however, JIT-SDP (including CP JIT-SDP) should be conducted using an online learning paradigm, where the classifiers continuously adapt and learn from new software changes over time (Cabral et al. 2019; Song et al. 2023b). Indeed, there has been a growing recognition in academia regarding the practicality and importance of studying JIT-SDP (and CP JIT-SDP) in the online scenario (Song et al. 2022; Song and Minku 2023; Song et al. 2023b; Cabral et al. 2019; Cabral and Minku 2022; Tabassum et al. 2022). In the online CP JIT-SDP setting, both OP and TP data are received as part of a continuous data stream. This sequential data stream can be used altogether to update the JIT-SDP model, ensuring their relevance and effectiveness as new information becomes available.

Recent study conducted by Tabassum et al. (2022) bridges the gap in online CP JIT-SDP by introducing three online CP methods: All-In-One (AIO), Filtering, and Ensemble. Before further discussing these methods, it is important to propose two fundamental concepts that serve as the basis for discussing their potential limitations. *Data diversity*, which is related to data distribution, refers to the observation that OP data does not resemble TP data and exhibits a broader range of data distribution. Including OP data with higher diversity in the training set can provide the Within-Project (WP) model (only use TP data) with a richer set of data information, thereby typically enhancing its predictive capability. Similarly, *data validity*, which is related to data similarity, emphasizes the importance of the requirement that OP data selected for training the model should be effective in improving the predictive

performance of JIT-SDP by being similar to the current TP data. A training set with a higher level of similarity to TP ensures that the learning process of prediction models does not deviate drastically, making the model more suitable for TP.

According to Tabassum et al. (2022), AIO incorporates all OP data into the model learning process, enabling the model to learn from a diverse range of defect-inducing changes and emphasizing the importance of data diversity. By considering a diverse set of training examples, the model gains insights from a wide spectrum of defect-inducing changes, which has the potential to enhance predictive performance of online CP JIT-SDP. In contrast, Filtering takes a different approach by selectively choosing from all OP data that exhibit similarities with the TP data, regardless of their original source projects. Instead of emphasizing data diversity, Filtering identifies and includes OP data that closely aligns with the specific characteristics of the TP. This selection process emphasizes the validity of data between OP and TP, ensuring that the model reflects the specific context and nature of the TP data. Tabassum et al. (2022) found that these data-level CP methods, specifically AIO and Filtering, exhibited promising and effective results in enhancing the predictive performance of online CP JIT-SDP – AIO leverages data diversity, while Filtering emphasizes data validity. However, the Ensemble method, which constructs separate predictors based on each OP and the TP data, has shown limited efficacy in enhancing predictive performance, as stated by Tabassum et al. (2022). This limitation may stem from the lack of consideration for the validity of OP data used to construct each individual OP learner, thereby hindering the potential for performance improvement in this online CP approach.

Both AIO and Filtering have their respective limitations. While AIO can benefit from high data diversity by incorporating all OP data across a wide range of data distribution, it may also include data from unrelated projects to the TP data, raising concerns about the validity of such data. Conversely, Filtering selects OP data that closely aligns with recent TP data, ensuring strong validity, but it also potentially results in a limited range of data distribution and insufficient diversity in the TP+OP data. These limitations can disrupt the balance between data diversity and data validity, which can impact the predictive performance of the JIT-SDP model.

To address these limitations and achieve a better balance between data diversity and validity, we aim to propose a novel online project-level CP approach. This approach considers project-level similarity to select *instructive* projects. An "instructive" project is an other project, which is of better trade-off between diversity and validity and thus has the potential in improving predictive performance of the target project for JIT-SDP at this period of time. That is,

$$Pf_{t+}(M_{T+I}) > Pf_t(M_T), T \sim I \tag{1}$$

where T is the target project, I is an instructive project. M_T is the model trained by data from T, M_{T+I} is the model trained by data from T and I. $Pf_t(M)$ is the predictive performance of model M at time step t (in each time step, there is the arrival of a test instance from T), and $t+$ is a time step after t. "$T \sim I$" means that T is similar to I at project-level. Compared to AIO, which may include unrelated and invalid OP data, this novel project-level approach which selects instructive projects would emphasize data validity by incorporating OP data that closely aligns with the TP. Compared to Filtering, the proposed CP approach would enrich the diversity of the chosen OP data by considering all software changes from the other projects that are similar to the TP. Additionally, previous studies have demonstrated the goodness of offline project-level CP approaches, indicating their potential applicability in the online scenario (Zhang et al. 2022; Kamei et al. 2016; Shehab et al. 2022; Cho et al.

2018). This, in turn, suggests that project-level CP approaches hold promise for improving online CP JIT-SDP.

The objective of this study is to design novel online project-level CP JIT-SDP approaches that leverage project-level similarities. The aim is to pursue a balance between data diversity and validity in the OP+TP data in methodology, addressing the limitations of existing online CP JIT-SDP methods. To achieve this objective, we address the following Research Questions (RQs):

RQ1 Are there project-level similarity metrics suitable for online JIT-SDP? How can we utilize these metrics to divide other projects that are instructive to the TP? To what extent can the collective project-level similarity help with selecting instructive other projects that improve the predictive performance of the WP over time?

RQ2 How can we propose a project-level CP approach that utilizes the project-level similarity metrics obtained in RQ1 to facilitate the predictive performance of WP? How does this novel method perform in comparison to state-of-the-art online CP methods?

RQ3 Building upon the project-level CP method proposed in RQ2, how can we further improve the predictive performance by incorporating multiple scales of OP selections? How does this extended method perform in comparison to state-of-the-art online CP methods?

To answer these RQs, we will conduct a thorough investigation of project-level similarity metrics used in existing CP JIT-SDP literature. We will carefully select appropriate project-level similarity metrics and based on our findings (answer to RQ1), we will propose an online project-level CP method called CroPS (Cross-Project method with Similarity). We will further propose an enhanced version called Multi-CroPS. To evaluate the effectiveness of our proposals, we will conduct systematic experiments based on 23 open-source GitHub projects. The main contributions of this paper are listed below:

- We conduct a thorough collection of project-level similarity metrics and evaluate their applicability and effectiveness in identifying instructive other projects for online CP JIT-SDP.
- Building upon the chosen project-level similarities, we propose CroPS, which can adaptively incorporate instructive other projects for training purposes and improve predictive performance of the WP. Experimental results demonstrate that CroPS can effectively improve the predictive performance while requiring lower computational costs compared to the latest data-level online CP methods.
- We propose Multi-CroPS, an ensemble-like framework, which incorporates multiple CroPS models with various OP threshold options. Experimental results significantly support the superiority of Multi-CroPS over other competing methods including our proposed CroPS. Despite its higher computational cost over CroPS, Multi-CroPS offers substantial performance improvement.
- We provide an analytical framework in terms of data diversity and validity for analyzing the limitation of existing online CP JIT-SDP methods. We highlight the importance of pursing a balance between data diversity and validity for online CP JIT-SDP.

The remainder of this paper is organized as follows. Section 2 discusses related work and presents motivating examples that show the potential advantages of adopting project-level CP methods. Section 3 proposes our project-level CP methods including CroPS and Multi-CroPS. Section 4 describes the 23 open-source projects investigated in this paper. Section 5 describes the design of our experiments, and our RQs are answered in Section 6. Threats to validity are discussed in Section 8 and the paper is concluded in Section 9.

2 Related Work and Motivating Examples

2.1 Online JIT-SDP

JIT-SDP operates in an online learning scenario, where software changes are continuously produced and labeled over time for prediction and training purposes, respectively (Tan et al. 2015; Song et al. 2023b). The availability of class labels, specified as defect-inducing and clean, experiences verification latency, where defect-inducing changes typically receive their labels with time delay and the labels of clean changes cannot be completely ensured. To address this verification latency issue, the waiting time method is employed in online JIT-SDP literature (Cabral et al. 2019; Tabassum et al. 2020; Song et al. 2022; Tabassum et al. 2022; Cabral and Minku 2022). Given a pre-defined waiting time, if the waiting time elapses without identifying any defect (or with the presence of a defect) associated with a software change, that change is labeled as clean (defect-inducing) for training purposes.

The issue of class imbalance frequently arises in online JIT-SDP, where there is a significant disparity between the number of defect-inducing changes (as the majority class) and clean changes (as the minority class). This class imbalance problem becomes particularly challenging in online scenarios where the overall data distribution is unknown in advance. To address the online class imbalance issue, Wang et al. (2015) proposed Oversampling Online Bagging (OOB) as a general learning machine. OOB has been widely adopted as a benchmark online JIT-SDP method (Cabral et al. 2019; Tabassum et al. 2020; Song et al. 2022; Tabassum et al. 2022; Cabral and Minku 2022). Subsequent to OOB, advanced online JIT-SDP methods have been developed that can deal with the class imbalance issue. These include Prediction-Based Sampling Adjustment (PBSA) (Cabral and Minku 2022) and Oversampling based Data Stream bagging with Confidence (ODaSC) (Song et al. 2022).

2.2 Offline CP JIT-SDP

Despite the fact that CP JIT-SDP should be operated in online scenarios, most of the existing studies on CP JIT-SDP focus on offline scenarios, where classifiers are constructed based on a pre-existing training set and remain unchanged thereafter (Zhang et al. 2022; Kamei et al. 2016; Shehab et al. 2022; Cho et al. 2018). This section provides a discussion of existing methods used for offline CP JIT-SDP. It is noteworthy that training a model solely with OP data may not effectively contribute to defect prediction of the TP data, as found by Fukushima et al. (2014) and we find that the success of CP JIT-SDP relies on the careful selection of instructive other projects that exhibit similarities with the target project as well as offer supplementary information.

To select instructive OP data, Kamei et al. (2016) introduced two types of project-level similarity metrics based on domain-agnostic and domain-aware. These metrics were used to identify instructive other projects for defect prediction on the TP data. Experimental results based on 11 open-source projects demonstrated that the methods employing both types of similarity to select instructive other projects for model training outperformed random selection. This indicates the effectiveness of utilizing project-level similarity metrics in enhancing the performance of CP JIT-SDP in offline scenarios. Cho et al. (2018) introduced other project-level similarity metrics by considering the number of developers who contributed to both sub-projects. They conducted experiments involving 5 projects consisting of 232 sub-projects. Experimental results demonstrated that the model trained across sub-projects exhibited better performance compared to model trained on a single sub-project.

They also found that selecting sub-projects with high similarity to build the model led to improved predictive performance. In a subsequent study, Shehab et al. (2022) introduced a cluster based method, which focused on grouping projects based on their shared libraries and functionalities. By identifying commonly used libraries among projects, they clustered together projects that exhibit a significant number of shared libraries. Experimental results based on 26 Apache Foundation projects demonstrated that constructing predictive models using projects from the same cluster as the target project yielded good predictive results. In another study, Zhang et al. (2022) proposed the Feature-based ENSEmble modeling method (FENSE). This method utilizes metrics to calculate the similarity, referred to as "transferability" in their paper, between the target project and other projects. It selects project models with the highest similarity and integrates them to predict the TP data. Extensive experiments involving 113,906 transferring pairs of 338 projects revealed that CP similarity is highly influenced by metrics such as programming language and the defect ratios of other projects.

There are additional studies that empirically investigated different CP JIT-SDP strategies. Yang et al. (2019) conducted a study comparing the performance of global and local models in the context of JIT-SDP. In their study, global models utilized all available data from both OP and TP for training, while local models employed a clustering approach to divide the combined OP+TP data into homogeneous regions, and trained a separate model for each region. In the local model, each test data is assigned to a specific region using a clustering method, and the corresponding region's model is used for prediction. Experimental results on 6 datasets showed that both global and local models have their advantages and disadvantages under different scenarios. This finding highlights the significance of considering the specific context of characteristics of the data when choosing between these approaches. Similarly, Lin et al. (2021) investigated the impact of combining data from different projects to build prediction models and its effect on model interpretation. Through their study of 20 open-source datasets, they found that models constructed by merging data while considering project similarity led to more interpretable models compared to a global approach that merges all available data. This suggests that researchers should take into account the similarity between projects when employing data merging techniques due to the potential goodness of interpretability of the resulting models.

Overall, existing offline CP JIT-SDP methods primarily depend on careful selection of instructive other projects. Studies have demonstrated encouraging results in improving predictive performance of TP data by selecting instructive other projects based on project-level similarity in offline scenarios.

2.3 Online CP JIT-SDP

Compared to offline scenarios, online JIT-SDP typically has limited availability of data to derive accurate similarity values between projects over time. This poses a significant challenge for directly adopting existing offline CP approaches, which often require a substantial amount of data to be effective. Online JIT-SDP is further complicated by verification latency and concept drift, which can impact the model training processes involved.

To fill this research gap, a recent study by Tabassum et al. (2022) introduced three online CP JIT-SDP methods: All-In-One (AIO), Filtering, and Ensemble. The AIO method combines all available OP data, together with the available training data from TP, to build a single model for online JIT-SDP. The Filtering method employs a filtering mechanism to select relevant data that are mixed with all OP and TP data for model training and prediction. It employs a fixed-size sliding window that maintains the most recent TP data. When OP data becomes

available, its Euclidean distance to the corresponding labeled TP data in the sliding window is calculated. If the average distance from the k most similar TP data falls below a specified threshold, the OP data meets the criteria for model updating and so is selected to update the model. Otherwise, the OP data is placed into a fixed-length queue for potential future use. Whenever a new TP data becomes available, Filtering checks if the discarded data in the queue meets the updating criteria and, if so, selects it for model updating. The Ensemble method combines multiple models trained on each individual OP or TP to enhance predictive performance. Given a testing TP data, all models in the ensemble make predictions, and the final prediction is determined based on the majority vote among the models.

Tabassum et al. (2022) examined 10 open-source projects and 9 private projects to investigate the predictive performance of those online CP methods. Experimental results demonstrated significant improvements when adopting AIO and Filtering in predictive performance compared to using only TP data. Specifically, the performance improvement in terms of G-Mean is up to 53.89%, 37.35%, and 29.03% during the initial period, sudden drop period, and stable period of software development, respectively. However, as discussed in Section 1, both AIO and Filtering may have their own limitations. AIO may incorporate data from unrelated projects, raising concerns about the validity of such data; whereas, Filtering may only select OP data within a limited range of data distribution, resulting in insufficient diversity. This means that there is still potential improvement for the online CP methods. Moreover, their experimental results showed that the Ensemble method cannot not yield satisfactory results in online scenarios. In fact, its performance was even inferior to using only TP data. The subpar performance of Ensemble can be attributed to the lack of validity of OP data that are used to construct each individual OP learner. For this reason, Ensemble will not be included in our study for a more effective online CP JIT-SDP comparisons.

Overall, state-of-the-art online CP JIT-SDP methods, such as AIO and Filtering, may still encounter the challenge of achieving a balance between diversity and validity. To resolve this issue and overcome the limitations of state-of-the-art online CP methods, there is a need for advanced online CP methods to effectively tackle the challenge, thereby improving the overall performance of online CP approaches.

2.4 Promising Potential of Online Project-level CP JIT-SDP

As discussed in Sections 2.1, 2.3, and 2.3, offline CP JIT-SDP methods may not be directly applicable to online scenarios due to the limited availability of TP+OP data required to calculate accurate similarity between projects over time. Conversely, state-of-the-art online CP methods, such as AIO and Filtering, encounter difficulties in achieving an optimal balance between diversity and validity. This paper aims to propose novel online project-level CP methods that adaptively select instructive OP data.

Our methods would involve accessing the project-level similarity between each other project and TP, promoting diversity by including all data from the chosen other project for learning purposes over time. Unlike AIO, which may include a large amount of unrelated and invalid OP data, our approach would emphasize data validity by considering other projects that exhibit high project-level similarity. Unlike Filtering, which may encounter insufficient diversity in the TP+OP data, our approach would take each OP as a basic unit and incorporate all available training data from the chosen OP for learning purposes, enriching the diversity. This idea draws inspiration from the promising outcomes achieved by offline CP methods. Our project-level OP selection is updated as part of the online CP JIT-SDP process, allowing the chosen other projects to vary over time based on the similarity between TP and other

projects. By doing so, our approach could potentially strive to achieve a more balanced combination of data diversity and validity, thereby enhancing the effectiveness of online CP JIT-SDP, including AIO and Filtering.

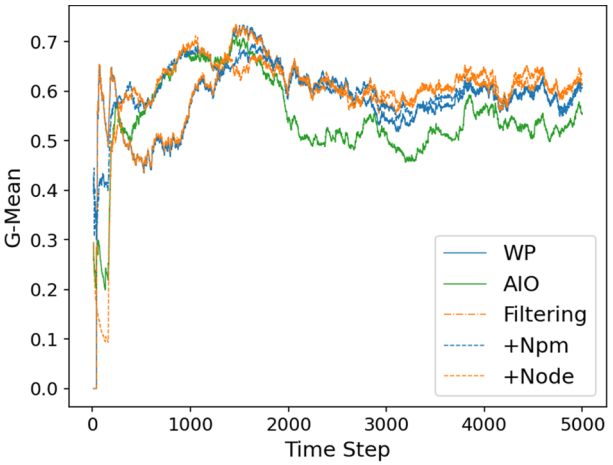
The rationale behind this project-level mechanism can be illustrated with the following motivating example. Let us consider a scenario where we want to perform online JIT-SDP on a TP, which lacks sufficient training examples in multiple prediction sessions. We have a other project dataset, OP, which can be utilized to augment the TP training set over time. Suppose the OP data is dissimilar to the TP data based on some data-level similarities, such as those employed in Tabassum et al. (2022). In this context, AIO would include all OP data in the training process, regardless of its irrelevance to TP. This indiscriminate inclusion of all OP data can have a negative impact on the predictive performance of WP due to the lack of CP *data validity*. On the other hand, Filtering would exclude all OP data for JIT-SDP due to their data-level dissimilarity, thereby failing to contribute to the prediction process in terms of enhancing *data diversity*.

However, if we can define a proper project-level similarity metric that takes into account higher-level factors such as programming languages and open-source licenses, the OP data can be considered "similar" to TP and thus utilized during specific training sessions. This similarity can stem from the fact that the two projects share the same open-source licenses, programming languages, or even by developers with similar experiences and expertise. If we neglected project-level information and solely focused on data-level information like AIO and Filtering do, we may overlook valuable instructive data, leading to either the inclusion or exclusion of all OP data for online JIT-SDP.

To validate our conjecture regarding the validity-vs-diversity balance, as discussed in Section 1, we will conduct an experimental study on state-of-the-art online CP methods, specifically AIO and Filtering. Experimental setup of this study is the same as the one described in Section 5.2. Moreover, we will explore the effectiveness of incorporating an entire OP dataset (project-level selection) for training each TP, aiming to investigate whether this mechanism could improve WP prediction. Specifically, for each TP, we include 22 other projects, with each OP representing all software changes from a single OP (more details about the datasets can be found in Section 4). To facilitate visualization, we only present the top two other projects yielding the best performance for each TP. This preliminary experimental study is referred to as *TP+1OP*. For example, "Bracket+Npm" represents that Bracket is the TP under investigation, and Npm is the incorporated OP.

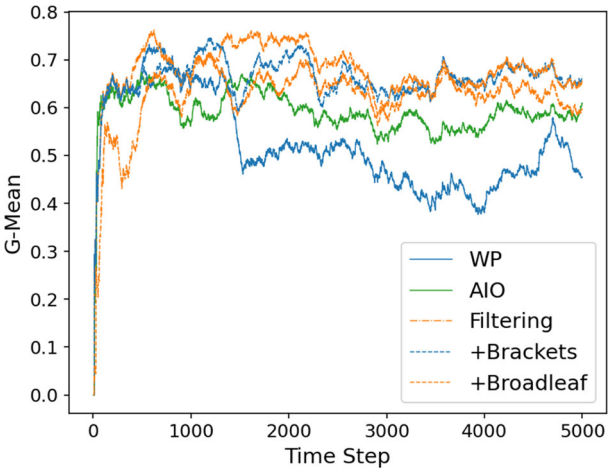
Figure 1 shows the online predictive performance of the investigated TP+1OP approach in comparison to AIO, Filtering, and the model constructed using only TP data on two representative datasets, namely Bracket and Neutron. The figures visualize the performance trends observed over the top 5,000 test steps. The corresponding tables provide an overall predictive performance on the Bracket and Neutron datasets. Predictive performance on other datasets are omitted due to space limitations, and their results can be found in the supplementary material for the sake of completeness.

We can see from Fig. 1a that both Bracket+Npm and Bracket+Node achieve slightly better overall predictive performance compared to the competitors. They unusually demonstrate remarkably competitive performance compared to the best-performing method across various test steps. This observation highlights the potential of incorporating an entire other project during specific prediction periods. Furthermore, AIO performs inferior to the WP method in terms of overall performance, though it performs better during time step 300-1,400. This suggests that including all cross projects without considering their validity to the target project may have a negative impact on predictive performance. Conversely, Filtering demonstrates



Method	G-Mean
+Npm	0.6026
+Node	0.6003
AIO	0.5491
Filtering	0.5965
WP	0.5826

(a) Bracket as the TP.



Method	G-Mean
+Bracket	0.6608
+Broadleaf	0.6578
AIO	0.5912
Filtering	0.6432
WP	0.5242

(b) Neutron as the TP.

Fig. 1 Comparison of predictive performance in terms of G-Mean over time on two representative datasets for online CP JIT-SDP approaches, including AIO and filtering. The benchmark performance is represented by "WP" using only TP data. The potential of project-level similarity approaches is explored by incorporating the entire dataset of a single other project in the learning process of WP, denoted as "+1OP". Overall performance across all test steps is reported in the accompanying tables

performance improvement for WP method, particularly after time step 2,700, indicating the potential benefit of including "proper" OP data during specific prediction periods.

We can also see from Fig. 1b that both Neutron+Bracket and Neutron+Broadleaf achieve better overall predictive performance compared to the competitors. They consistently outperform the other methods across the majority of test steps. Furthermore, Neutron+Bracket performs better than Neutron+Broadleaf during the time steps 0-1300. While Neutron+Broadleaf outperforms Neutron+Bracket during the time steps 1,300-2,800. This phenomenon suggests that the OP that is best suited to the target project may vary over

time. Consequently, the project-level similarity between other projects and TP would also vary over time, depending on the current status of TP data. This finding also suggests the importance of adaptively deciding the project-level similarity between other projects and TP over time.

Based on these experimental results, we can conclude that the proposal of the project-level CP approach, which incorporates adaptive selection based on specific project-level similarity, holds potential in resolving the issues of current online CP methods. The findings indicate that the indiscriminate inclusion of all other projects, as AIO does, or the exclusion of other projects based on data-level dissimilarity, as Filtering does, can have unhelpful or even negative impacts on predictive performance. Conversely, the proposal of adopting specific project-level CP approach takes into account higher-level factors, such as programming language used and the licenses, with the goal of incorporating instructive OP data while ensuring the validity of the selective other projects.

3 Methodology

3.1 Project-level Similarity Metrics for Online CP JIT-SDP

This section addresses RQ1 by performing a thorough literature review for the collection of project-level similarity metrics. Subsequently, we calculate our project-level similarity using the selected metrics as a means of RQ1.

3.1.1 Collection of Project-level Similarity

In this section, we collect project-level similarity metrics from a wide range of academic papers focusing on the topic of CP JIT-SDP (Zhang et al. 2022; Kamei et al. 2016; Lin et al. 2021; Zheng et al. 2021). From this collection, we selectively choose the metrics that are applicable to online scenarios.

Kamei et al. (2016) employed categorical metrics such as company, audience, user interface, use-database, and programming language to describe a project that can be used to quantify similarity between projects. They also calculated Spearman correlation between predictive labels and feature values as an additional numerical similarity metric. Lin et al. (2021) considered numerical metrics such as lines of code, numbers of project files, numbers of commits, and numbers of developers to capture project's characteristics. Zheng et al. (2021) employed the Jensen-Shannon (JS) to measure the similarity between projects. Later, Zhang et al. (2022) summarized four types of similarity metrics between projects: model-level, project-level, social-level, and technical-level. Model-level metrics include factors such as the number of commits, defect ratio, and predictive performance. Project-level metrics primarily focus on domain-aware metrics such as project popularity, owner type, license, language, and textual similarity. Social-level metrics relate to developer and include the number of core developers, external contributors, and collaborators across different projects. Technical-level metrics consider code quantity and dependencies, which depict the package relationships between projects. In addition, a widely recognized paper in the domain of general SDP by Zimmermann et al. (2009) is also included in the review of similarity metrics due to its prominence, which utilized the median, maximum, and standard deviation of individual features adopted in their prediction model as similarity metrics.

It is worth noting that similarity metrics we previously collected were primarily designed for offline scenarios, where some of them were decided based on the full information available at the completion of the project development. Specifically, certain metrics, like the textual similarity proposed by Zhang et al. (2022), require the collection of project descriptions and `ReadMe` files from two projects, which can only be obtained when these projects have been completed. This renders such metrics impractical for online CP JIT-SDP. Consequently, it is necessary to exclude those similarity metrics that are not applicable to online scenarios from our study. Among the similarity metrics we have collected, 20 metrics are identified to be applicable to online scenarios.

Table 1 presents an overview of the similarity metrics being applicable to online scenarios. The first 13 metrics are inherent to the project itself that provide insights into various aspects of the projects under investigation and can be collected at the beginning of the project development. The last 7 metrics require calculations over time based on available TP+OP data. Notably, three metrics including “OS", “Audience", and “Owner Type" have been excluded from this study because of their high consistency observed in their values across investigated projects, for which at least 22 projects share the same values in these metrics. As a result, 17 metrics are retained for the formulation of our project-level similarity as they are suited for online CP JIT-SDP.

Also, Table 2 presents the domain-aware metric values for the projects investigated in this paper[1]. These metric values were collected from GitHub[2]. Domain-aware metrics can be obtained even at the initial phase of the project development process and remain static. Conversely, the values of data-based metrics depend on the current status of software project development and tend to change over time. Therefore, we cannot include these values in the table. The values marked as “NA" in the table indicates that they are not accessible from GitHub, so we have to handle their values manually. We spare categorical metrics marked as “NA" as an additional distinct value. This approach ensures that the presence of “NA" in categorical metrics does not impact the distance calculation based on this categorical feature of other values. For numerical metrics marked as “NA", we replace them with the average value of the corresponding metric across other projects. This substitution ensures that the absence of numeric metric values does not disrupt the calculation process.

3.1.2 Formulation of Project-level Similarity

Based on the chosen similarity metrics, various calculations can be performed to quantify the similarity between projects as discussed in Section 2.2. For example, Kamei et al. (2016) determined similarity between projects exclusively using categorical metrics, such as the programming language used. Zhang et al. (2022) employed Euclidean distance to calculate similarity between projects mostly based on numerical metrics such as the number of commits. Zheng et al. (2021) adopted the JS divergence to quantify the project-level similarity.

Previous similarity calculations have respected either part of categorical or part of numerical metrics, without fully leveraging the most informative aspects of the metrics to develop a unified formulation. Therefore, our objective is to incorporate the maximum available information by formatting the similarity calculation between projects using a combination of categorical and numerical metrics, as listed in Table 1. By doing so, we can create a more thorough and informative representation of similarity between projects that is specifically tailored to the context of online CP JIT-SDP.

[1] The description of the datasets will be presented in Section 4.

[2] The information was collected on December 29, 2022

Table 1 An overview of the collective project-level similarity metrics

Metrics	Type	Describe	Source
Starting time	N	The starting year	Zhang et al. (2022)
License	C	The open-source licenses	Zhang et al. (2022)
Language	C	Programming languages used	Zhang et al. (2022); Kamei et al. (2016); Zimmermann et al. (2009); Lin et al. (2021)
Core dev	N	The number of core members	Lin et al. (2021); Zimmermann et al. (2009)
Domain	C	The main domain	Zimmermann et al. (2009)
Company	C	The company responsible for the project	Kamei et al. (2016); Zimmermann et al. (2009)
	C	Whether the project is built for interaction with end users or built for development professionals	Kamei et al. (2016); Zimmermann et al. (2009); Lin et al. (2021)
	C	Whether a project runs only on Windows or on multiple operating systems	Zimmermann et al. (2009)
User interface	C	The type of user interaction	Kamei et al. (2016); Zimmermann et al. (2009); Lin et al. (2021)
Database	C	Whether or not the project persists data using a database	Kamei et al. (2016); Zimmermann et al. (2009); Lin et al. (2021)
Local	C	Whether the project is available in one language or is available in several international languages	Zimmermann et al. (2009)
Single Pl	C	Whether the project uses only one programming language	Zimmermann et al. (2009)
	C	Owner type of the project	Zhang et al. (2022)
N_commits	N	Total number of commits	Zhang et al. (2022); Lin et al. (2021)
Defect ratio	N	The ratio of defect commit	Zhang et al. (2022)
Median	N	Median of the features used in the prediction model	Zimmermann et al. (2009)
Maximum	N	Maximum of the features used in the prediction model	Zimmermann et al. (2009)
Standard deviation	N	Standard deviation of the features used in the prediction model	Zimmermann et al. (2009)
Spearman correlation	N	Correlation between a dependent variable and each predictor variable	Kamei et al. (2016)
JS divergence	N	A measurement of the degree of difference between two projects	Zheng et al. (2021)

Out of the total 20 metrics, the first 13 metrics are domain-aware as they are inherent to the project itself, providing insights into various aspects of the projects under investigation. The last 7 metrics are data-based and require calculations over time based on available TP+OP data. The metric type is denoted by “N" for numerical metrics and “C" for categorical metrics. Notably, the metrics marked with strike-through have been excluded from our study. This decision was made due to the high consistency observed in their values across investigated projects, where at least 22 projects share the same values in OS, Audience, and Owner Type. In the end, a total of 17 metrics are retained for the formulation of our project-level similarity in this study

Table 2 Values of the domain-aware metrics of the projects studied in this paper

Project	Starting time	License	Language	Core Dev	Domain	Company	User interface	Data- base	Local	Single Pl
Ansible	2012	GPL-3.0	Python	64	Automatic operation and maintenance	Ansible	Toolkit	0	0	0
Brackets	2011	MIT	JavaScript HTML	203	Web	Adobe	Graphical	0	1	0
Broadleaf	2008	NA	Java	7	E-commerce platform	Broadleaf	Non-interactive	1	0	0
Camel	2007	Apache-2.0	Java	1200	Program develop	apache	Non-interactive	1	0	0
Corefx	2014	MIT	C#	560	Cross-platform develop	microsoft	Non-interactive	1	1	0
Django	2005	BSD-3-Clause	Python	64	Web	django	Toolkit	1	1	0
Elasticsearch	2010	NA	Java	144	Search engine	elastic	Graphical	1	1	0
Fabric	2011	Apache-2.0	Go	67	Web	Hyperledger	Non-interactive	1	0	1
Googleflutter	2014	BSD-3-Clause	Dart	85	Mobile application UI Toolkit	flutter	Toolkit	0	0	0
Homebrew	2017	BSD-2-Clause	Ruby	48	Package manager	Homebrew	Toolkit	0	0	0
jGroups	2003	Apache-2.0	Java	1	Java library	None	Toolkit	0	0	1
Neutron	2011	Apache-2.0	Python	NA	Cloud compute	openstack	Toolkit	0	0	1
Node	2009	NA	JavaScript C++,Python	310	JavaScript runtime	nodejs	Non-interactive	0	0	0
Nova	2010	Apache-2.0	Python	NA	Cloud compute	openstack	Toolkit	0	0	0
Npm	2009	NA	JavaScript	10	Package manager	npm	Graphical	1	1	1
Panda	2009	BSD-3-Clause	Python	37	Data analysis	pandas	Toolkit	0	0	0
Pytorch	2012	NA	C++,Python	52	Machine Learning	pytorch	Toolkit	0	0	0
Rails	2004	MIT	Ruby	57	Web	rails	Non-interactive	1	1	0
Rust	2010	Apache-2.0 MIT	Rust	177	Programming language	rust	Non-interactive	0	0	0
Tensorflow	2015	Apache-2.0	C++,Python	252	Machine Learning	tensorflow	Toolkit	0	0	0
Tomcat	2006	Apache-2.0	Java	1200	Web	apache	Non-interactive	1	0	0
VScode	2015	MIT	TypeScript	4400	Code editor	microsoft	Graphical	1	1	0
wp-Calypso	2014	GPL-2.0	JavaScript TypeScript	192	Web	Automattic	Graphical	1	0	0

We have excluded three metrics (OS, audience, and owner type) due to their high level of consistency. The presence of “NA" means that the corresponding information is not available from GitHub

In particular, each categorical metric in Table 1 has been reformulated in a dummy format, following the approach employed by Kamei et al. (2016). To calculate the similarity between projects based on these metrics, the Euclidean measure is utilized. To prevent the dominance of any single metric, a normalization process has been applied. Each similarity metric is normalized within the range of [0, 1] using the min-max normalizer. In the end, our formulation for the similarity measurement $\mathcal{S}_{AB}$ between projects A and B is

$$\mathcal{S}_{AB} = \frac{1}{\sum_{k=1}^{||\mathcal{C}||} |cm_{k,A} - cm_{k,B}| + \sum_{k=1}^{||\mathcal{N}||} |nm_{k,A} - nm_{k,B}| + js_{AB} + 1} \tag{2}$$

where $cm_{k,A}$ ($cm_{k,B}$) represents the value of the k-th categorical similarity metric of project A (B), $nm_{k,A}$ ($nm_{k,B}$) represents the value of the k-th numerical similarity metric of project A (B), and $||\mathcal{C}||$ ($||\mathcal{N}||$) represents the total number of categorical (numerical) metrics employed. The term js_{AB} denotes the JS divergence between projects A and B, where larger values indicate higher difference between the projects. The addition of one unit in the denominator serves to avoid dividing by zero and to ensure resulting similarity value falling within (0, 1], providing interpretable similarity values.

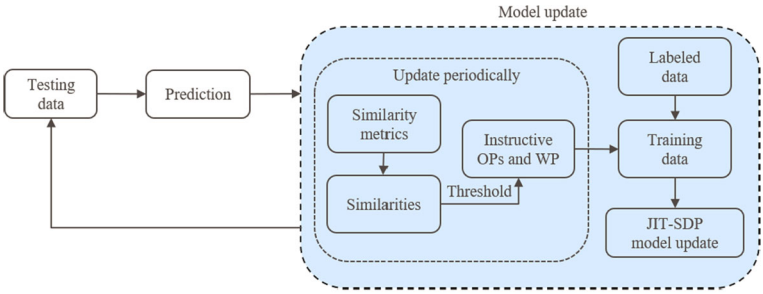
Experimental results of RQ1, as will be discussed shortly in Section 6.1, demonstrate the effectiveness of these similarity metrics in well capturing the quantification of similarity between projects to a certain degree. Thus, there a great promise in extending the application those metrics to the online scenario where they can be used for online OP selection.

3.2 CroPS: Cross-project Method with Similarity

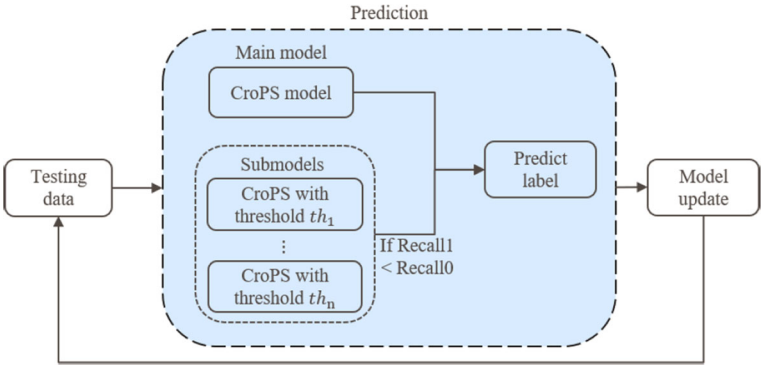
This section addresses RQ2 by proposing an online project-level CP approach called CroPS (Cross-Project method with Similarity). CroPS is developed based on the selective similarity metrics in Table 1 and the similarity formulation between projects in (2). The flow chart of CroPS is outlined in Fig. 2(a) and the procedure is detailed in Algorithm 1.

As outlined in Fig. 2(a), the proposed CroPS operates in an online learning scenario with verification latency, where the training and testing data continuously arrive with time. When a new testing TP data arrives, the latest JIT-SDP model is used to make the prediction (Line 6 of Algorithm 1). Due to verification latency inherent in JIT-SDP, the true label of this software change cannot be obtained immediately after its prediction, distinguishing it from the conventional online learning scenario (Tan et al. 2015). To account for verification latency, CroPS stores these "unlabeled" data into a cache window, awaiting their true labels (Line 18). Once the true label of a defect-inducing software change is obtained within the designated waiting time period, a defect-inducing training data is created for training purposes (Line 21-22). Instead, if a change remains non-defect-inducing after a given waiting time w, a clean training data is created for training purposes (Line 28-29). In case where a change is initially produced as a clean training data and is subsequently found to be defect-inducing, a second training data with the true label of defect-inducing is produced and used for model update (Line 38-41). These learning procedures align with previous online JIT-SDP with verification latency (Tabassum et al. 2022; Song et al. 2023b).

We now focus on elucidating the core part of CroPS, which is the "Model update" module depicted in Fig. 2(a). The objective of this module is to identify the instructive other projects for the current TP data and incorporate all available training examples from these other projects into the TP train process. This module consists of three main components: 1) monitoring the project-level similarity metrics over time, 2) identifying instructive other



(a) Flow chart of CroPS with a focus on the "Model update" part. The implementation details can be found in Algorithm 1.



(b) Flow chart of Multi-CroPS with a focus on the "Prediction" part. The "Model update" part remains consistent with CroPS. The implementation detail can be found in Algorithm 2.

Fig. 2 Flow chart illustrating the "test-then-train" process of online scenarios in CroPS and Multi-CroPS. These flow charts provide a visual representation of the sequential steps involved in the prediction and adaptation of CP JIT-SDP learning processes

projects for additional training data, and 3) updating the model. Each of these components will be discussed in the following paragraphs.

To monitor the project-level similarity metrics over time, we maintain sliding windows for each project, including both TP and other projects, with a pre-defined size of w_{crops} (e.g., 500 software changes). These sliding windows are initialized at the start of each project (Line 2 of Algorithm 1). The project-level metrics in Table 1 are continuously updated as new training data associated with the respective project become available over time. This is to ensure that we have the most up-to-date similarity metrics between projects, which can be used to calculate the project-level similarities to identify instructive other projects over time. These project-level similarities are updated after every arrival of t_{up} TP data using (2) (Line 7-9 of Algorithm 1). While it is technically feasible to update the similarities whenever new labeled training data arrives, such a frequent similarity calculation would lead to high computational costs. Moreover, considering that a small change in the access to the new labeled TP+OP data is unlikely to induce a significant impact on the calculation of the similarity metrics, it is more efficient to update the similarity periodically rather than for every data arrival. If practitioners can afford the high computational cost of such update, one can make this by assigning the model parameter t_{up} as 1.

To select instructive other projects for the training purposes, we use a window to maintain the chosen other projects. This window is initialized at the beginning of the JIT-SDP process (Line 3). To make the selection, a simple threshold method is utilized (Line 10-14). other projects with similarity values exceeding θ_{crops} are included in the TP training process until their similarities drop below θ_{crops}. The selection of threshold θ_{crops} plays a crucial role as it determines the extent of other projects chosen. A larger threshold leads to more other projects being selected for TP model training, which can encourage the diversity but impact the validity of OP data incorporated into the TP model. Conversely, a lower threshold results in limited other projects being selected for TP model training, which can ensure the validity but confine the diversity of OP data incorporated. Therefore, it is important to carefully choose the threshold parameter to strike a balance between the diversity and validity of the data obtained from the selected other projects.

To update the TP model, we employ the newly available training examples from both the TP and the selected other projects. This process is carried out in lines 24-26, 32-34, 42-44 of Algorithm 1. This test-then-train procedure is repeated iteratively until the completion of the target project's development.

This section proposed a novel online project-level CP method called CroPS, which relies on a threshold strategy to select instructive OP data throughout the development of a software project to make a balance between data diversity and validity. However, the presence of concept drift in online JIT-SDP (McIntosh and Kamei 2018) poses a challenge in maintaining the effectiveness of this threshold-dependent method. Continuously assigning appropriate thresholds over time is important to ensure optimal performance. This burden of continuous threshold assignment may discourage developers from adopting our method. Hyper-parameter tuning can be employed to adaptively decide thresholds over time. But when and how to decide these thresholds in time is non-trivial and poses high computational load. In the subsequent section, we propose an advanced multi-threshold CroPS method to mitigate the issue faced by the threshold-dependent CroPS method.

3.3 Multi-CroPS: Multi-threshold CroPS

To address the issue of adaptively tuning the threshold in CroPS, we propose an ensemble-like framework named Multi-threshold CroPS (Multi-CroPS). This approach maintains multiple CroPS models to integrate various OP-choosing thresholds, while preserving the high performance of online JIT-SDP. Notably, the learning framework of Multi-CroPS deliberately incorporates the domain knowledge that defect prediction is typically more challenging than clean prediction due to class imbalance. The flow chart of Multi-CroPS is depicted in Fig. 2(b), where the training procedure for each individual model is the same as that of CroPS, as depicted in Fig. 2(a). The crucial part of Multi-CroPS is the "Prediction" module, responsible for producing final predictions by integrating the predictions from all base CroPS models. The subsequent paragraphs explain the key points of the prediction procedures. Detailed procedures can be found in Algorithm 2.

As illustrated in Fig. 2(b), the main-vs-sub modeling strategy is employed in Multi-CroPS. The main CroPS model serves as the primary prediction reference for the test data, and the remaining CroPS models under different thresholds operate as sub-models. During the prediction phase, Multi-CroPS keeps monitoring the predictive performance of the online CP model. If the model demonstrates improved performance on the defect-inducing class (Recall 1 $\geq$ Recall 0), Multi-CroPS uses the predicted label from the main model as the final label, as shown in Line 4-5 of Algorithm 2. However, if the model exhibits better performance

Algorithm 1 Cross-project method with similarity (CroPS).

Input: DS = stream of incoming changes from several projects, b = index identifying the target project, w_{crops} = sliding window size, t_{up} = update period, θ_{crops} = select threshold

1: Initialize predictive model m
2: Initialize sliding windows $slid_p$ with size w_{crops} for each project p
3: Initialize a selected OP window OP_{used} contains all OP and TP
4: **for** each incoming software change $x_p^t \in DS$ **do**
5: **if** $p = b$ **then**
6: $\widehat{y} \leftarrow$ predict(m, x_p^t) // x_p^t denotes a change arriving from project p at time step t of the project
7: **if** $t \ mod \ t_{up} = 0$ **then**
8: **for** each OP c **do**
9: Calculate $sim_{c,p}$ using the data stored in $slid_c$ and $slid_p$ as (2)
10: **if** $sim_{c,p} \geq \theta_{crops}$ **then**
11: Add c into OP_{used}
12: **else**
13: Remove c from OP_{used}
14: **end if**
15: **end for**
16: **end if**
17: **end if**
18: Store x_p^t in a cache window WFL // WFL is a window of incoming examples waiting to be used for training

19: **for** each item q^i in WFL **do**
20: $ind \leftarrow$ index identifying the project which contains q^i
21: **if** a defect was linked to q^i at a time step $\leq t$ **then**
22: Create a defect-inducing training example for q^i
23: Add $training\_example$ into $slid_{ind}$
24: **if** $ind \in OP_{used}$ **then**
25: train(m,$training\_example$)
26: **end if**
27: **else**
28: **if** q^i is older than w **then**
29: Create a clean $training\_example$ for q^i
30: Add $training\_example$ into $slid_{ind}$
31: store q^i in CL // CL is a window of clean training examples
32: **if** $ind \in OP_{used}$ **then**
33: train(m, $training\_example$)
34: **end if**
35: **end if**
36: **end if**
37: **end for**

38: **if** a defect was linked to a training example in CL before time t **then**
39: $ind \leftarrow$ index identifying the project contains q^i
40: Swap label of $training\_example$ to defect-inducing
41: Remove $trainingexample$ from CL
42: **if** $ind \in OP_{used}$ **then**
43: train(m,$training\_example$)
44: **end if**
45: **end if**
46: **end for**

on the clean class (Recall 1 < Recall 0), Multi-CroPS also considers the predicted labels provided by the sub-models (Line 7-13). In an aggressive manner, if there is at least one prediction label among these models that indicates a defect-inducing class (label 1), Multi-CroPS sets the final predicted label as 1.

The main-vs-sub model strategy is adopted in Multi-CroPS for several reasons. CroPS models with different thresholds select OP data at varying scales. Typically, a CroPS model with a higher threshold usually exhibits lower diversity but higher validity, whereas a model with a lower threshold usually demonstrates higher diversity but lower validity. The main model is designed to strike a better trade-off between validity and diversity at the first place. It aims to potentially outperform other CroPS models in specific JIT-SDP scenarios. However, the optimal CroPS model is unknown in advance and can even change over time, making it almost impossible to determine in the online learning scenario. By selecting the most promising CroPS with a good balance of validity and diversity as the main model, and utilizing others as sub-models to aid in the defect prediction, Multi-CroPS can better simulate the varying selection strategies the most suited to different online JIT-SDP scenarios. This approach allows for the incorporation of various CroPS models in determining the predicted label of a test software change. It provides a more effective alternative to treating each CroPS equally, as typically adopted in the conventional ensemble learning approaches.

Another important aspect of the Multi-CroPS approach is the incorporation of domain knowledge regarding the challenges faced by JIT-SDP models in achieving good Recall 1 (Recall of defect-inducing changes) compared to Recall 0 (Recall of clean changes) (Cabral et al. 2019; Song et al. 2022; Cabral and Minku 2022). This discrepancy can be attributed to factors such as class imbalance (defect-inducing changes occurring less frequently than clean changes) and label noise resulting from verification latency (defect-inducing changes initially being labeled as clean changes for model updating). To address this challenge, Multi-CroPS adopts an aggressive combination strategy. If at least one CroPS model predicts the label of class 1 (defect-inducing), the final prediction label is set as class 1. This defect-intending strategy is only employed when it is observed that the prediction model performs better on clean changes (Recall 1 $<$ Recall 0). This can prevent a decline in performance resulting from an over-excessive focus on the defect-inducing class.

It is noteworthy that, using the true labels to calculate the true performance of the model at the current time is unattainable due to verification latency, as the true labels for the most recent data are not yet available. To address this issue, we use the estimated performance based on surrogate time steps and observed labels (Song and Minku 2023; Song et al. 2023b). Specifically, when need to access the recall 0 and recall 1 of the model, we do not utilize the recently arrived test data. Instead, we leverage the recently arrived training data, for which the observed labels have already been received. By comparing the observed labels of the recently arrived training data with the model's prediction labels when the data was previously used as test data, we derive the estimated recall 0 and recall 1 metrics. The waiting time used in our experiments is 15 days, which is recommended by Song and Minku (2023). Although there may be discrepancies between the estimated performance and the true performance, the estimated recall 0 and recall 1 can effectively reflect the model's recent performance according to the conclusions drawn in (Song et al. 2023a). By employing these estimates, we can assess whether the model has been sufficiently attentive to defect changes, thereby meeting the requirements of Multi-CroPS.

Overall, Multi-CroPS offers the potential to improve defect prediction compared to the base CroPS model by utilizing diverse information from other projects of varying scales. It also places a dedication on the importance of defect-inducing changes. However, it should be noted that this performance improvement entails additional computational costs and resources required to maintain multiple CroPS models. Table 20 provides the average running time in seconds required to complete one commit, covering the prediction and model updating processes. Related discussion will be provided in Section 6.3.1.

Algorithm 2 Multi-threshold CroPS (Multi-CroPS).

Input: DS_{tp} = stream of incoming changes from target projects, M_{crops} = set of crops models with different select threshold θ_{crops}, θ_{main} = the θ_{crops} of the main crops model in M_{crops}

$m_{main} \leftarrow$ the crops model in M_{crops} with $\theta_{crops} = \theta_{main}$
for each incoming change $x^t \in DS_{tp}$ **do**
  $Recall\ 1, Recall\ 0 \leftarrow$ Evaluate the performance of m_{main}
  if $Recall\ 1 \geq Recall\ 0$ **then**
    $\widehat{y} \leftarrow$ predict(m, x^t) // x^t is a change arriving at time step t
  else
    $y_{temp} \leftarrow$ predict(m, x^t)
    for each model m_i in M_{crops} **do**
      if predict(m_i, x^t)=1 **then**
        $y_{temp} \leftarrow 1$
      end if
    end for
    $\widehat{y} \leftarrow y_{temp}$
  end if
  update models in M_{crops} as in Algorithm 1
end for

4 Datasets

We evaluate our proposed methods (and the adopted project-level similarity metrics) based on 23 open-source GitHub projects, among which 9 (Brackets, Broadleaf, Camel, Fabric, jGroups, Neutron, Nova, Npm, and Tomcat) were made available by Cabral et al. (2019), 7 (Corefx, Django, Rails, Rust, Tensorflow, VScode, and wp-Calypso) were made available by Song et al. (2022), and the remaining 7 projects were collected by ourselves. We choose these projects deliberately as they have more than 2 years of duration, rich history (>7k commits) and a wide range of defect-inducing change ratios (1%∼51%). An overview of these projects is provided in Table 3.

The `Commit Guru` tool (Rosen et al. 2015) was used to extract historical commits of those projects in Table 3, including labels and input features of software changes. The tool is based on the SZZ algorithm (Śliwerski et al. 2005) to decide the label of a software change, which is defect-inducing (class 1) or clean (class 0). As SZZ has been known to suffer from label noise (Kim et al. 2008; Nugroho et al. 2020; Rodríguez-Pérez et al. 2022; Rezk et al. 2022; Herbold et al. 2022), we conducted a manual inspection of a random sample of software changes from each project. This allows us to evaluate the data quality when applying the SZZ technique to determine the labels of the software changes in each project investigated in this paper.

Following the same procedures in Song et al. (2023a), we invited four experts with at least 4 years of programming experience to label those randomly chosen changes for each individual project. They worked in pairs to discuss each of the changes and collaboratively decide its label. This human annotation process generated two independent sets of human-generated labels (Pair 1 and Pair 2). We follow the labeling process of Herbold et al. (2022) and McIntosh and Kamei (2018) to annotate software changes as defect-fixing or non-defect-fixing. This manual inspection process can reflect the level of noise induced by using SZZ to identify software changes as defect-fixing and non-defect-fixing. As SZZ uses fixes to identify defect-inducing software changes, a high level of noise in the SZZ identification of changes as defect-fixing and non-defect-fixing means a high level of noise in the SZZ labels of defect-inducing and clean.

Table 3 Summary of the datasets investigated in this work

Projects	Total changes	%Defect- inducing	Time period	Main language
Ansible	42,208	35.49	06-02-2012 - 26-07-2020	Python
Brackets	11,477	33.88	08-12-2011 - 08-12-2017	JavaScript
Broadleaf	12,034	20.54	20-12-2008 - 22-12-2017	Java
Camel	29,860	20.80	19-03-2007 - 07-12-2017	Java
Corefx	26,191	6.91	07-11-2014 - 01-11-2019	C#
Django	25,662	42.16	13-07-2005 - 28-09-2019	Python
Elasticsearch	45,223	35.15	08-02-2010 - 27-07-2020	Java
Fabric	12,282	20.74	14-04-2011 - 06-12-2017	Go
Googleflutter	17,564	1.28	24-10-2014 - 27-07-2020	Dart
Homebrew	43,694	2.12	23-09-2017 - 29-03-2019	Ruby
jGroups	17,947	17.52	09-09-2003 - 05-12-2017	Java
Neutron	8,347	39.73	01-01-2011 - 24-12-2017	Python
Node	26,432	33.22	16-02-2009 - 25-07-2020	JavaScript
Nova	22,872	43.52	28-05-2010 - 23-01-2018	Python
Npm	7,544	17.88	30-09-2009 - 28-11-2017	JavaScript
Panda	18,991	51.21	05-08-2009 - 24-07-2020	Python
Pytorch	27,195	40.12	25-01-2012 - 27-07-2020	C++
Rails	56,049	25.37	24-11-2004 - 27-09-2019	Ruby
Rust	68,344	2.03	17-06-2010 - 27-10-2019	Rust
Tensorflow	64,135	24.88	07-11-2015 - 01-01-2020	C++
Tomcat	18,559	27.93	27-03-2006 - 07-12-2017	Java
VScode	51,459	2.27	13-11-2015 - 26-10-2019	TypeScript
wp-Calypso	30,015	22.53	29-05-2014 - 31-10-2019	JavaScript

"%Defect-inducing" represents the percentage of defect-inducing software changes over the total number of extracted ones

For each project, we randomly select 100 changes (50 defect-fixing and 50 non-defect-fixing) for human annotation. Both the commit message and the associated issue were used to label the commit, unless the commit message lacked an issue ID within it. In particular, if a commit message addressed an issue identified by a given ID that corresponded to a defect, this change was labeled as defect-fixing. It is worth noting that this manual inspection process did not account for the noise arising from `git blame` within `Commit Guru` (Bludau and Pretschner 2022; Rezk et al. 2022).

The Kappa scores (Cohen 1960) (1) between SZZ and Pair 1, (2) between SZZ and Pair 2, and (3) between Pair 1 and Pair 2 are reported in Table 4. Following Hall et al. (2014), we interpret Kappa scores as follows: $[-1, 0]$ – less than chance agreement, [0.01, 0.20] – slight agreement, [0.21, 0.40] fair agreement; [0.41, 0.60] moderate agreement; [0.61, 0.80] substantial agreement, and [0.81, 0.99] – almost perfect agreement. We can see from Table 4 that the Kappa scores (1) and (2) indicate at least moderate agreement for all datasets. For ten datasets including Ansible, Broadleaf, Corefx, Fabric, Homebrew, Neutron, Npm, Panda, Pytorch, and Tomcat, the average values of (1) and (2) indicate a substantial agreement; for another four datasets including Django, Tensorflow, VScode, and wp-Calypso, those Kappa values indicate almost perfect agreement.

Table 4 Kappa scores

Dataset	(1) SZZ vs Pair 1	(2) SZZ vs Pair 2	Average of (1) and (2)	(3) Pair 1 vs Pair 2
Ansible	0.640 (82)	0.740 (87)	0.690 (84.5)	0.704 (85)
Brackets	0.538 (81)	0.472 (78)	0.505 (79.5)	0.708 (87)
Broadleaf	0.710 (90)	0.652 (88)	0.681 (89.0)	0.787 (92)
Camel	0.438 (78)	0.560 (84)	0.499 (81.0)	0.633 (84)
Corefx	0.757 (89)	0.629 (83)	0.693 (86.0)	0.684 (86)
Django	0.779 (89)	0.874 (94)	0.827 (91.5)	0.739 (87)
Elasticsearch	0.480 (74)	0.420 (71)	0.450 (72.5)	0.491 (75)
Fabric	0.642 (88)	0.806 (94)	0.724 (91.0)	0.662 (88)
Googleflutter	0.540 (77)	0.600 (80)	0.570 (78.5)	0.604 (81)
Homebrew	0.720 (86)	0.760 (88)	0.740 (87.0)	0.722 (86)
jGroups	0.507 (86)	0.530 (87)	0.519 (86.5)	0.905 (97)
Neutron	0.720 (86)	0.780 (89)	0.750 (87.5)	0.622 (81)
Node	0.580 (79)	0.560 (78)	0.570 (78.5)	0.537 (77)
Nova	0.555 (80)	0.564 (80)	0.560 (80.0)	0.709 (86)
Npm	0.680 (84)	0.620 (81)	0.650 (82.5)	0.541 (79)
Panda	0.700 (85)	0.700 (85)	0.700 (85.0)	0.794 (90)
Pytorch	0.560 (78)	0.760 (88)	0.660 (83.0)	0.542 (76)
Rails	0.432 (79)	0.556 (82)	0.494 (80.5)	0.726 (89)
Rust	0.594 (88)	0.480 (83)	0.537 (85.5)	0.480 (83)
Tensorflow	0.858 (96)	0.805 (94)	0.832 (95.0)	0.797 (94)
Tomcat	0.574 (81)	0.684 (86)	0.629 (83.5)	0.745 (89)
VScode	0.875 (95)	0.810 (92)	0.843 (93.5)	0.875 (95)
wp-Calypso	0.875 (96)	0.775 (92)	0.825 (94.0)	0.710 (90)
Median	0.640 (85)	0.652 (86)	0.660 (85.0)	0.708 (86)
Average	0.641 (84.7)	0.658 (85.4)	0.650 (85.0)	0.683 (86)

The values in "()" are the number of agrees. That is, out of the 100 changes under the same dataset, the number of results annotated by both parties agree

Moreover, the median and average Kappa scores across datasets between human annotators and SZZ (average of (1) and (2)), and between human annotators themselves (3) are also reported in Table 4. They both indicate a substantial agreement, showing that the level of agreement between human annotators and SZZ is in line with the level of agreement between humans themselves. This suggests that the labels provided by SZZ were in general unlikely to be worse than the labels given by humans.

The input features of software changes consist of 14 metrics that can be grouped into 5 dimensions as (1) diffusion: NS (number of modified subsystems), ND (number of modified directories), NF (number of modified files) and Entropy (distribution of modified code across each file), (2) size: LA (lines of code added), LD (lines of code deleted) and LT (lines of code in a file before the change), (3) purpose: FIX (whether or not the change is to fix a defect), (4) history: NDEV (number of developers that changed the modified files), AGE (average time interval between the last and the current change) and NUC (number of unique changes to the modified files), and (5) experience: EXP (developer experience), REXP (recent developer experience) and SEXP (developer experience on a subsystem). These metrics have shown to be good indicators for JIT-SDP (Kamei et al. 2013) and have been widely used in online JIT-SDP (Cabral et al. 2019; Cabral and Minku 2022; Song et al. 2022; Tabassum et al. 2022).

Prior studies have also recommended to preprocess these 14 features for better predictive performance in JIT-SDP (Kamei et al. 2013; Yang et al. 2019). We follow Kamei et al. (2013) to conduct the same preprocessing procedures and ultimately obtain 12 transformed feature metrics in this study. The main preprocessing procedures are:

1. *Removing highly correlated features*: (i) LA and LD are normalized by dividing by LT. (ii) LT and NUC are normalized by dividing by NF since LT and NUC are highly correlated with NF. (iii) ND and REXP are removed as they are highly correlated with NF and EXP, respectively.
2. *Dealing with skew*: Since most features are highly skewed, each metric went through logarithmic transformation except for FIX (a Boolean variable).

5 Experimental Setup

This section presents the experimental setup for answering the three research questions proposed in Section 1. The source code for this work will be made available as open-source at GitHub and Zenodo.

5.1 Base JIT-SDP Models

Online CP approaches should integrate with base JIT-SDP methods to enable defect prediction. This study opts for OOB (Wang et al. 2015), ODaSC (Song et al. 2022), and PBSA (Cabral and Minku 2022) as the base methods, considering their state-of-the-art status and popular adoption in online JIT-SDP. These three base online JIT-SDP models are also taken as the WP methods and are included as part of our competing methods.

OOB is designed to address the online imbalance issue and has been widely adopted in prior studies of online JIT-SDP (Cabral et al. 2019; Tabassum et al. 2020; Song et al. 2022; Tabassum et al. 2022; Cabral and Minku 2022). ODaSC is proposed to mitigate the impact of one-sided label noise resulting from verification latency of JIT-SDP. It employs cluster-based calculations to determine the confidence of labels of each newly labeled training example. When a clean labeled training data exhibits to have a higher confidence level on its being defect-inducing compared to defect-inducing labeled training examples, this new training data would be more likely as defect-inducing and used for model updating. PBSA is built upon Oversampling Rate Boosting (ORB) (Cabral et al. 2019) to cater for the frequent concept drift in JIT-SDP. ORB automatically adjusts the resampling ratio of each class to adapt to the changing class imbalance. PBSA further incorporates prediction results into the automatic adjustment of the resampling ratio. By considering the model's prediction labels, PBSA fine-tunes the resampling ratio to facilitate the model's recovery from concept drift.

We employ grid search for parameter tuning of the base models for each project. A base model is constructed using a specific parameter combination to make online predictions on data from the first 1,000 time steps of each project. The parameter combination that yields the highest performance (measured by G-Mean) will be selected as the using set of parameters for that project. We set candidate parameters according to their corresponding studies (Song et al. 2022; Cabral and Minku 2022; Tabassum et al. 2022). Hyper-parameter candidate values of base JIT-SDP models are summarized in Table 5. ODaSC, PBSA and OOB have some common parameters: ensemble size (n_tree) and decay factor of imbalance tracer (θ_{imb}). The specific parameter for ODaSC is: decay factor of clusters (θ_{cl}). The specific parameters for PBSA are: growth of boosting factors (m), target value (th), and allowed deviation (p).

Table 5 Hyper-parameter values for tuning the base JIT-SDP models

Name	Parameters	Candidate values
OOB	n_tree	{5,10,15,20,30}
	θ_{imb}	{0.9, 0.95, 0.99, 0.999}
ODaSC	n_tree	{5,10,15,20,30}
	θ_{imb}	{0.9, 0.95, 0.99, 0.999}
	θ_{cl}	{0.8, 0.9}
PBSA	n_tree	{5,10,15,20,30}
	θ_{imb}	{0.9, 0.95, 0.99, 0.999}
	m	{1.5, 2.0, e}
	th	{0.2, 0.3, 0.4, 0.5, 0.6}
	p	{0.15, 0.25, 0.35}

5.2 Experimental Setup for RQ1

To complete our answer to RQ1, we need to evaluate whether our designed similarities can effectively reflect the instructiveness between projects. To construct the ground truth, we design the "TP+1OP" experimentation, where we only incorporate the data from a single OP into the training set for the learning purposes of the TP model. By evaluating the predictive performance of the TP model using different OP data, we can determine whether the included OP that is decided based on our proposed project-level similarity in Section 3.1 would provides instructive information to the TP model.

Specifically, we treat each individual project as the TP and conduct the "TP+1OP" experiments using each of other projects as the single other project. These "TP+1OP" experiments utilize software changes from both the TP and the OP, which are available prior to the 5000th test step of the TP. To account for experimental randomness, we repeat each "TP+1OP" combination 20 times, getting a more reliable ground truth. We then determine whether a OP is instructive to the TP by examining the impact in the predictive performance when incorporating OP data in the model. If the inclusion of OP data leads to an increase (decrease) in the predictive performance of the model, we consider the OP as instructive (uninstructive) to the TP, being the ground truth. On the other side, we calculate the similarities between other projects and the TP using (2), considering data availability before the 5000th test step of the TP. Ultimately, we can evaluate whether these calculated similarities can effectively reflect the underlying ground truth.

To perform the experimental investigation, we frame this task as a binary classification task, where instructive other projects are labeled as class "1" (indicating that combining the OP data into TP would lead to an improved performance for the model), while others are labeled as class "-1" (representing a worse performance after combining the OP data into the TP). The similarity values between each individual other project and the given TP serve as predictive scores. Higher similarity values indicate a greater probability that the OP is instructive to the TP. To evaluate the performance of our project-level similarity, we utilize the AUC (Area Under Curve) metric, which represents the probability that a positive example is ranked higher than a negative example in predictions. We calculate AUC values, area under ROC curve, using the predictive scores (similarities) and the class labels obtained from the ground truth. In the ROC curve, each point corresponds to a decision threshold (classification threshold that assigns labels to an input changeset based on the inferred probability of the classifier). The AUC value is computed while the decision threshold varies from 0 to 1 (Obuchowski

2005). When evaluating the performance of our proposed project-level similarity, we use the calculated project-level similarity as the inferred probability of the classifier. For the projects with higher similarities than the decision threshold, the prediction labels are "1". The prediction labels and the true labels which are get from the ground truth are used to calculate the TPR and FPR at the decision threshold. We move the decision thresholds between 0 and 1 and calculate the TPR and FPR under each decision threshold to finally form the ROC curve and calculate the AUC value.

We conduct a comparison between random selection and the selection method using our proposed project-level similarity metrics. The goal is to investigate and empirically demonstrate the effectiveness of our project-level similarity metrics in comparison to a naive random selection approach. Notably, only after justifying the validity of our proposed similarity-based methods in RQ1, we would then proceed to apply such similarity in RQ2 to further investigate its impact on predictive performance on the models. Regarding the competing method of random selection, its inferred probability is randomly generated within the range of 0 to 1. Consequently, when we change the decision threshold (classification threshold) from 0 to 1, the theoretical AUC value will consistently be 0.5. Therefore, the random selection method has an AUC value of 0.5. If the AUC for a TP (calculated by our proposed project-level similarity metrics) is larger than 0.5 (where 0.5 corresponds to random selection), we consider the derived similarities to effectively represent the instructiveness between other projects and the TP, in accordance with the guidelines proposed by Fawcett (2006). To determine whether our approach of selecting other projects significantly outperforms random selection, we employ a one-tailed and paired Wilcoxon Signed rank test across all projects (Woolson 2007). This statistical tests allow us to assess the significance of the differences observed and evaluate the superiority of our OP selection method compared to random selection.

We also investigate whether the calculated similarities can be improved by employing a metric selection method, leading to better selection of the other projects. To this end, we employ the conventional genetic algorithm (Mirjalili and Mirjalili 2019). In the metric selection process, we represent each individual as a vector, where each index corresponds to a project-level metric. The value at each index ranges from 0 to 1. If the value is equal to or greater than 0.5, we include the corresponding metric in the similarity calculation. For each individual, we calculate the similarity using the selected metrics. We then compute the AUC value based on these similarity, which serves as the fitness value for the individual. To evaluate the effectiveness of the metric selection process, we employ a one-tailed and paired Wilcoxon Signed rank test. This statistical test helps us determine whether the use of metric selection leads to improved performance compared to not using it. Experimental investigation results will be shortly discussed in Section 6.1.

5.3 Experimental Setup for RQ2 and RQ3

To answer RQ2 and RQ3, we need to compare our approaches against other state-of-the-art online CP methods, including AIO and Filtering (Tabassum et al. 2022). AIO involves incorporating all OP data for model updating without the use of any parameters. On the other hand, Filtering selects OP data that is similar to the current TP data, and it utilizes certain parameters in the process. We employ grid search to perform parameter tuning for the CP methods for each project. A CP method is constructed using a specific combination of parameters to make online predictions on the first 1,000 time steps of each project. The parameter combination that yields the highest performance in terms of G-Mean, is selected as the chosen parameter setting for that project. It is important to note that the parameters

Table 6 Hyper-parameter candidate values for the competing CP approaches

Name	Parameters	Candidate values
Filtering	$w_{filtering}$	{500, 700, 1000}
	k	{50, 100, 200}
	max_dist	{0.6, 0.7, 0.8}
	$discard\_size$	{500, 1000}
CroPS	w_{crops}	{500, 700, 1000}
	t_{up}	{100, 200, 500}
	θ_{crops}	{0.1, 0.15, 0.2, 0.25, 0.3, 0.5}

of the CP methods varies for different JIT-SDP methods. Since we had 23 projects and 3 base models, this will result in a total number of 69 parameter combinations, so that the performance could be maximized for the specific context of each experimental scenario.

Candidate values of the hyper-parameters for each CP method are summarized in Table 6. For CroPS, the parameters include the sliding window size (w_{crops}), the update period (t_{up}), and the select threshold (θ_{crops}). In particular, considering that both CroPS and Filtering have the size of sliding window as an important tuning parameter, we follow Tabassum et al. (2022) to set the candidate values as {500, 700, 1000}, following their setup for sensitivity analysis, to ensure a fair and credible comparison between the proposed CroPS and Filtering. For Multi-CroPS, the parameters of the sub-models use the same values as CroPS. We set 6 candidates for θ_{crops}, and thus Multi-CroPS would have one main CroPS model with the optimal θ_{crops} obtained from CroPS, and 5 sub-models with other values of θ_{crops}. AIO does not have any hyper-parameter. Filtering has parameters including the sliding window size ($w_{filtering}$), the number of top short distances (k), the distance threshold for similarity (max_dist), and the maximum size of the queue of dissimilar OP instances ($discard\_size$). We use the candidate values for Filtering as described in Tabassum et al. (2022).

We use Geometric Mean of Recall 0 and Recall 1 (G-Mean) (Kubat et al. 1998) and Matthews Correlation Coefficient (MCC) (Matthews 1975) to evaluate predictive performance of online CP models investigated in this paper. We use tp to denote true positives (the number of defect-inducing software changes that are predicted correctly), fn to denote false negatives (the number of defect-inducing software changes that are erroneously predicted as clean), tn to denote true negatives (the number of clean software changes that are predicted correctly) and fp to denote false positives (the number of clean software changes that are erroneously predicted as defect-inducing). Based on them, G-Mean $= \sqrt{\frac{tp}{tp+fn} \cdot \frac{tn}{tn+fp}} \in [0, 1]$, MCC $= \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp+fp) \cdot (tp+fn) \cdot (tn+fp) \cdot (tn+fn)}} \in [-1, 1]$. Different from accuracy or precision, G-Mean is known to be robust against class imbalance, which is particularly important for studies suffering from class imbalance such as JIT-SDP (Cabral et al. 2019; Wang et al. 2018). MCC is adopted as it has become popular in the area of JIT-SDP (Chicco and Jurman 2020; Chicco et al. 2021).

To answer RQ2 and RQ3, we use the average retrospective performance to evaluate different CP methods (Song et al. 2023b). The retrospective performance of a JIT-SDP model at time step u is computed using the following equation:

$$E_{cn}(u) = \theta \cdot E_{cn}(u-1) + (1-\theta) \cdot ||\widehat{y_u} - y_u||_G \tag{3}$$

where $E_{cn}(u-1)$ denotes the retrospective performance at time step $u-1$, $\widehat{y_u}$ and y_u denote the predictive and true labels of the test data X_u, respectively. The norm $||\widehat{y_u} - y_u||_G$ measures

the correctness of the prediction made by the JIT-SDP model, and the symbol G represents different performance metrics, such as G-Mean or MCC. The forgetting factor, denoted by $\theta \in (0, 1)$, controls the emphasis placed on past evaluation examples compared to new ones. In this study, we set $\theta = 0.99$, following previous work (Song et al. 2022). We list the process of how G-Mean is calculated. If X_u belongs to class i, $Recall_u^i = \theta \cdot Recall_{u-1}^i + (1-\theta) \cdot \mathbb{I}_{\widehat{y_u}=i}$, where the class i is zero (clean) or one (defect-inducing), u is the current time step, and $\mathbb{I}_{\widehat{y_u}=i}$ is the indicator function, which evaluates to one if $\widehat{y_u} = i$ and to zero if otherwise. If X_u does not belong to class i, $Recall_u^i = Recall_{u-1}^i$. G-Mean at the current time step can be calculated as $\sqrt{Recall_u^1 \times Recall_u^0}$. In a word, the average retrospective performance across all test steps is used to evaluate and compare the performance of online CP methods for JIT-SDP.

Table 7 RQ1 – Performance comparison between our project-level similarity and the benchmark random selection method in the selection of instructive other projects in terms of AUC

Project	ODaSC	OOB	PBSA
Ansible	0.688	0.608	0.717
Brackets	0.763	0.768	0.844
Broadleaf	0.523	0.533	0.468
Camel	0.656	0.667	0.618
Corefx	0.333	0.367	0.698
Django	0.811	0.690	0.842
Elasticsearch	0.580	0.508	0.708
Fabric	0.676	0.720	0.509
Googleflutter	0.500	0.909	N/A
Homebrew	N/A	N/A	N/A
Jgroups	0.545	0.608	0.658
Neutron	0.395	0.524	0.522
Node	0.789	0.629	0.786
Nova	0.423	0.675	0.662
npm	0.500	0.447	0.362
Panda	0.508	0.723	0.783
Pytorch	0.886	0.470	0.500
Rails	0.500	N/A	0.500
Rust	0.444	0.477	0.408
Tensorflow	0.508	0.538	0.692
Tomcat	0.589	0.490	0.378
Vscode	0.408	0.539	0.553
wp-calypso	0.540	0.383	0.617
p-value	0.027	0.006	0.002

AUC values great than 0.5 (the AUC value derived from the random selection method) indicate that our project-level similarities for selecting instructive other projects is superior to the random selection. The designation "N/A" signifies that all other projects are instructive, thus the division into instructive or uninstructive categories are not applicable. The "p-value" is calculated using one-tailed and paired Wilcoxon Signed rank test, comparing the AUC values to 0.5 (representing the AUC value derived from the random selection method).If the "p-value" is smaller than 0.05, it indicates that the adoption of our similarity metrics for other project selection is significantly superior to random selection

Comparisons between online CP JIT-SDP methods are conducted based on the mean predictive performance across 20 runs to account of the stochasticity of our experiments. We report the result corresponding to the waiting time of 15 days, following previous related studies which usually delivers good predictive performance (Song et al. 2022, 2023a). The double Scott-Knott ESD tests (Tantithamthavorn et al. 2018) is performed for statistical comparison between JIT-SDP methods across all projects, with which the ranks of these methods can be computed. Methods with the same ranks imply no significant difference in their performance, whereas distinct ranks indicate that the method with a higher rank exhibits significantly superior performance compared to the others. Friedman tests (Demšar 2006) with Conover post hoc tests are performed for statistical comparison between JIT-SDP methods on each project. By applying these tests to each project individually, we can identify the method that achieved the highest rank and exhibited statistically significant difference from other methods. We refer to this method as the "best performed method" for each project. It is important to note that there may be multiple "best performed methods", that is, these methods have significantly better performance than other methods, and there are no significant differences between them. Meanwhile, there may be no "best performed method", that is, there is no method have significantly better performance than other methods.

Table 8 RQ1 – Project-level similarity between each other project and rust as the TP

Projects	Similarity	Rank
Ansible	0.240	19
Brackets	0.325	7
Broadleaf	0.437	6
Camel	0.259	15
Corefx	0.296	11
Django	0.204	23
Elasticsearch	0.488	2
Fabric	0.323	8
Googleflutter	0.302	10
Homebrew	0.283	13
jGroups	0.226	21
Neutron	0.242	18
Node	0.454	5
Nova	0.246	16
Npm	0.466	3
Panda	0.243	17
Pytorch	0.461	4
Rails	0.224	22
Rust	1.000	1
Tensorflow	0.296	12
Tomcat	0.264	14
VScode	0.232	20
wp-Calypso	0.304	9

"Rank" reports the ranks of similarity of this other project to Rust according to our similarities. Uninstructive other projects on all base models are highlighted in yellow

6 Experimental Results

6.1 RQ1: Effectiveness of our Project-level Similarity Metrics

This section aims to investigate the effectiveness of the project-level similarity metrics presented in Section 3.1, completing our answer to RQ1. The experimental setup was explained in Section 5.2. Table 7 presents the experimental results in terms of AUC for each TP with the three base JIT-SDP models. The p-values obtained for each base method are 0.027, 0.006, and 0.002, respectively, which provide a clear observation that the adoption of our proposed similarities for other project selection leads to significantly better results compared to random selection. Therefore, the incorporation of these project-level similarity metrics can benefit in selecting instructive other projects, thereby demonstrating the effectiveness of our project-level similarity proposal.

An interesting case in Table 7 is that when adopt Rust as the TP, all base models exhibit AUC values below the threshold of 0.5, showing inferior performance of our similarity metric to the benchmark random OP selection. To investigate the potential factors contributing to such low AUC for Rust, we conduct an analysis by comparing the derived similarity values

Table 9 RQ1 – Experimental results to show the effectiveness of similarity metrics selection

Project	ODaSC	OOB	PBSA
Ansible	0.991	0.983	0.975
Brackets	1.000	0.942	1.000
Broadleaf	0.954	0.829	0.786
Camel	1.000	1.000	1.000
Corefx	0.850	0.867	0.889
Django	0.978	1.000	0.987
Elasticsearch	0.902	0.856	0.908
Fabric	0.814	0.917	0.848
Googleflutter	1.000	1.000	N/A
Homebrew	N/A	N/A	N/A
jGroups	1.000	0.804	0.842
Neutron	0.757	0.988	0.811
Node	0.875	0.864	0.955
Nova	0.938	0.897	0.862
Npm	0.863	0.727	0.900
Panda	0.908	0.892	0.950
Pytorch	1.000	0.902	0.895
Rails	1.000	N/A	1.000
Rust	0.833	0.970	0.817
Tensorflow	1.000	0.838	0.858
Tomcat	0.967	0.931	0.889
VScode	0.785	0.934	0.789
wp-Calypso	0.960	0.742	0.958
p-value	0.000	0.000	0.000

The context is similar as Table 7. The only difference is that metrics selection is used here

with the ground truth obtained with the procedures depicted in Section 5.2. Based on the identified ground truth, multiple other projects consisting of Broadleaf, Camel, Django, Elasticsearch, Fabric, Neutron, Node, Npm, and Panda were found. These other projects were consistently identified as uninstructive to Rust across all base models. Table 8 further presents the similarities between Rust and these other projects, along with their corresponding ranks. other projects highlighted in yellow indicate that they were uninstructive to Rust across all base models. Checking the ranks of these uninstructive other projects, we can observe certain inconsistencies. Broadleaf, Elasticsearch, Fabric, Node, and Npm ranked as 6th, 2nd, 8th, 5th, and 3rd, respectively, which are unreasonable given their lack of instructiveness. Consequently, these inconsistencies in terms of project-level similarities may contribute to the low AUC values for Rust.

Moreover, we conduct an additional experiment to examine the potential benefits of the automatic selection of similarity metrics for online JIT-SDP. Experimental results regarding the metric selection are presented in Table 9. Their p-values are smaller than 0.0001, indicating a significant performance improvement in terms of AUC resulting from the metric selection. However, it is important to note that there is no universally applicable set of similarity metrics that can effectively cover all other project environments. Our experiments showed that the metrics that are the most suited to a specific project and base model may not be able to yield satisfactory results when applied to a different project or base model. The optimal metrics set may also vary across different time intervals, and periodic updating of the metric set may incur substantial computational costs. Consequently, for the investigation conducted on RQ2 and RQ3, we have decided not to adopt the metrics selection. However, it is noteworthy that if an organization is rich in computational resources, dynamic metric selection is recommended as it can encourage valuable benefits in better predictive performance.

Table 10 The number of times each similarity metric is selected in the metrics selection results

Metric	Times
User interface	33
Spearman correlation	30
Company	27
Local	27
Core dev	24
Median	24
License	21
Language	18
Single Pl	18
Starting time	15
Max	15
Standard deviation	12
Domain	9
Defect ratio	9
JS divergency	9
Database	6
N_commit	6

The more times, the more important the metric is. Similarity metrics selection has a total of 64 sets of results, and its effectiveness is show in Table 9

Even though there is no universally applicable set of similarity metrics, we provide the number of times each similarity metric is selected in Table 10, to suggest the importance of each metric. We can see that the most 4 important metric are “User Interface", “Spearman Correlation", “Company" and “Local". “User Interface" is the type of user interaction. In general, software with a graphical interface requires more complex UI design. Projects that have graphical interface may be more similar to each other than projects that do not. “Spearman Correlation" is calculated using recent data from a project. It represents the correlation between the change features and the labels in the project, which is an important metric to determine whether projects are similar. “Company" represents the company responsible for the project. Projects from the same company are more likely to be similar than projects from other companies. “Local" indicates whether the software has been localized for different languages. In general, localized projects employ translation functions. Projects that are localized are more likely to be similar to each other. Therefore, if developers have obstacles in collecting similarity metrics, we recommend that they prioritize adopting “User Interface", “Spearman Correlation", “Company" and “Local", which are relatively more important.

Answer to RQ1: We conducted a thorough literature review for the collection of project-level similarity metrics. Based on the selection of these metrics, we formulated our project-level similarity measure. This measure allows us to quantify the instructiveness of each individual OP with the TP. Experimental results demonstrated the effectiveness of these similarity metrics in well capturing the degree of similarity between projects. Therefore, they show great promise for online OP selection. Automated selection of similarity metrics based on genetic algorithms cannot provide a universally optimal set of similarity metrics and thus is not recommended for practical usage when the organization has limited computational resources and domain knowledge. If developers have obstacles in collecting similarity metrics, “User Interface", “Spearman Correlation", “Company" and “Local" are recommended first, which are relatively more important.

6.2 RQ2: Predictive Performance of CroPS

Based on our project-level similarity metrics, we propose CroPS to select instructive other projects for online CP JIT-SDP. This section aims to investigate whether and to what extent CroPS can improve predictive performance compared to both WP methods and other online CP methods. Experimental results are presented in Tables 11, 12, 13, 14, 15, and 16 in terms of G-Mean and MCC using different base JIT-SDP methods depicted in Section 5.1, individually. The row labeled “Ranking(RQ2)" in the tables represents the rankings of the competing methods, determined using the double Scott-Knott ESD test.

As we can see from those tables, the experimental results consistently show that CroPS outperforms the WP methods across all base models in terms of all performance metrics, showing the effectiveness of our proposed online CP method. While CroPS does not always exhibit significant advantages over online CP methods including AIO and Filtering. It performs significantly better than or similar to AIO and Filtering in most datasets, except for when PBSA is used as the base model and G-Mean is the evaluation metric.

To gain a deeper understanding of the advantages and potential limitations of CroPS, we choose to analyse the projects that exhibit the best and worse performance improvement ratios in terms of G-Mean when comparing WP to CroPS. This analyses allow us to explore the factors contributing to various performance differences between the WP methods and

Table 11 RQ2 & RQ3 – Average values and standard deviations of G-mean on ODaSC

Project	WP		AIO		Filtering		Crops		Multi-Crops	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
Ansible	0.641	0.002	0.634	0.003	**0.642**	0.002	0.636	0.002	**0.642**	0.002
Brackets	0.616	0.009	0.609	0.006	0.614	0.006	**0.625**	0.011	**0.625**	0.011
Broadleaf	0.620	0.005	0.569	0.006	0.630	0.003	**0.652**	0.003	0.651	0.003
Camel	0.663	0.004	0.641	0.003	0.668	0.003	0.666	0.004	**0.682**	0.002
Corefx	0.472	0.015	0.628	0.007	0.441	0.012	0.506	0.011	0.631	0.007
Django	0.691	0.004	0.683	0.001	**0.697**	0.002	0.683	0.002	0.681	0.003
Elasticsearch	0.689	0.002	0.592	0.004	0.691	0.002	**0.699**	0.003	0.693	0.004
Fabric	0.657	0.004	0.634	0.004	0.666	0.003	**0.685**	0.004	**0.685**	0.004
Googleflutter	0.160	0.019	0.589	0.013	0.262	0.021	0.589	0.013	**0.598**	0.016
Homebrew	0.475	0.008	0.425	0.011	0.477	0.006	0.402	0.015	**0.600**	0.008
jGroups	0.553	0.002	0.558	0.004	0.556	0.004	0.556	0.004	**0.564**	0.004
Neutron	0.587	0.014	**0.694**	0.005	0.640	0.005	0.660	0.010	0.660	0.010
Node	0.630	0.005	0.639	0.004	0.647	0.005	0.633	0.004	**0.665**	0.003
Nova	0.587	0.011	0.638	0.008	0.633	0.005	**0.655**	0.004	0.654	0.004
Npm	0.528	0.005	0.628	0.007	0.527	0.010	0.630	0.006	0.629	0.006
Panda	0.648	0.012	0.631	0.005	**0.660**	0.007	0.648	0.006	0.648	0.006
Pytorch	0.672	0.003	0.655	0.003	0.669	0.002	**0.689**	0.001	0.688	0.002
Rails	0.537	0.008	0.626	0.006	0.575	0.008	0.599	0.008	**0.665**	0.002
Rust	0.330	0.008	0.557	0.005	0.323	0.010	0.383	0.008	**0.594**	0.005
Tensorflow	0.695	0.003	0.659	0.003	0.694	0.002	0.691	0.003	0.688	0.004
Tomcat	**0.618**	0.004	0.548	0.007	0.608	0.003	0.608	0.002	0.613	0.003
VScode	0.486	0.005	0.596	0.007	0.503	0.008	0.596	0.007	**0.649**	0.004
wp-Calypso	0.661	0.003	0.609	0.007	0.662	0.003	0.624	0.006	0.623	0.006
Ranking(RQ2)	2		2		1		1		–	
Ranking(RQ3)	4		4		3		2		1	

Here shows the average values (Avg) and standard deviations (Std) of G-Mean across all time steps and 20 runs. ODaSC is the base JIT-SDP model. The bold Avg values represent the "best performed method" for each project, which has the significantly better results than other methods. The Friedman test and Conover post hoc test are used on the results of 20 runs to get the "best performed method". There may be multiple "best performed methods", or there may be no "best performed method" on each project. "Ranking(RQ2)" shows the rank of each CP methods studied in RQ2. "Ranking(RQ3)" shows the rank of each CP methods studied in RQ3. The rank is calculated by double Scott-Knott ESD test

our CroPS. These analyses would be conducted using the base method ODaSC for it usually getting the best predictive performance against other base JIT-SDP models with respect to different datasets and in terms of different performance metrics. The discussions are provided in the subsequent two paragraphs.

As shown in Table 11, Googleflutter exhibits the highest improvement ratio from WP to CroPS, with an improvement ratio of 266.9%. It is worth noting that Googleflutter is a highly imbalanced project, with a low proportion of defect-inducing changes at only 1.28 (see Table 3). Figure 3 shows the online performance of Googleflutter and we can observe that the first defect-inducing change is predicted at approximately time step 2,000, indicating the detrimental effect of class imbalance on the predictive ability of the model, particularly

Table 12 RQ2 & RQ3 – average values and standard deviations of MCC on ODaSC

Project	WP		AIO		Filtering		Crops		Multi-Crops	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
Ansible	0.316	0.003	0.311	0.004	0.311	0.004	**0.326**	0.003	0.324	0.003
Brackets	0.278	0.013	0.265	0.008	0.267	0.011	**0.285**	0.013	**0.285**	0.013
Broadleaf	0.278	0.006	0.300	0.006	0.280	0.005	0.322	0.006	0.321	0.006
Camel	0.355	0.006	0.321	0.004	0.360	0.004	0.361	0.004	**0.371**	0.005
Corefx	0.261	0.011	0.278	0.012	0.217	0.009	0.263	0.007	0.278	0.014
Django	0.418	0.004	0.397	0.002	0.417	0.004	0.405	0.004	0.400	0.004
Elasticsearch	0.412	0.004	0.347	0.004	0.417	0.003	**0.424**	0.005	0.407	0.007
Fabric	0.339	0.007	0.367	0.007	0.362	0.006	**0.386**	0.007	**0.386**	0.007
Googleflutter	0.080	0.015	0.187	0.024	0.133	0.015	0.187	0.024	**0.207**	0.031
Homebrew	0.267	0.007	0.254	0.008	0.274	0.006	0.242	0.012	**0.351**	0.009
jGroups	0.192	0.004	0.209	0.007	0.207	0.006	0.193	0.006	0.180	0.007
Neutron	0.236	0.014	**0.407**	0.009	0.302	0.007	0.360	0.016	0.360	0.016
Node	0.324	0.006	0.316	0.005	0.337	0.004	0.330	0.005	**0.340**	0.005
Nova	0.273	0.010	0.323	0.010	0.301	0.006	**0.337**	0.006	0.336	0.006
Npm	0.166	0.006	0.275	0.014	0.162	0.015	0.277	0.012	0.277	0.012
Panda	0.339	0.013	0.325	0.008	0.345	0.012	0.323	0.010	0.323	0.010
Pytorch	0.365	0.005	0.369	0.004	0.365	0.003	**0.392**	0.003	0.391	0.003
Rails	0.263	0.007	0.319	0.007	0.290	0.006	0.298	0.006	**0.344**	0.003
Rust	0.191	0.006	0.243	0.008	0.175	0.009	0.227	0.006	**0.270**	0.008
Tensorflow	0.397	0.004	0.377	0.003	0.395	0.004	0.391	0.006	0.388	0.006
Tomcat	**0.268**	0.008	0.249	0.008	0.260	0.006	0.262	0.005	0.246	0.005
VScode	0.299	0.005	0.316	0.008	0.305	0.007	0.316	0.008	**0.357**	0.008
wp-Calypso	0.353	0.006	0.313	0.007	0.349	0.006	0.324	0.006	0.321	0.006
Ranking(RQ2)	3		2		2		1		–	
Ranking(RQ3)	2		2		2		1		1	

Please refer to Table 11 for more description

concerning the defect-inducing class. In the case of WP, its Recall 1 remains consistently low probably due to the scarcity of defect-inducing changes, resulting in a lower G-Mean. The Filtering method attempts to address this imbalance issue by incorporating similar OP data into the model updating process. But the improvement in Recall 1 is limited, possibly due to the difficulty in identifying similar OP defect-inducing changes due to their scarcity of this class. In contrast, CroPS and AIO adopt a different strategy. By adopting a low threshold, CroPS can include all OP data, similar to AIO. Despite the limited number of defect-inducing changes in Googleflutter, both AIO and CroPS are able to enhance their datasets by incorporating OP defect-inducing changes, effectively compensating for the scarcity. As a result, they achieve higher Recall 1 compared to WP and Filtering, leading to superior G-Mean.

As shown in Table 11, Homebrew exhibits the worst performance improvement from WP to CroPS, even with a decrease of 15.3%, showing that CroPS has a negative impact on WP in the case of Homebrew. Similar to Googleflutter that owns the highest performance improvement, Homebrew is also an imbalanced project, with a low proportion of defect-inducing class at only 2.12 (see Table 3). Figure 3 shows that CroPS performs better in Recall 1 than WP

Table 13 RQ2 & RQ3 – Average values and standard deviations of G-Mean on OOB

Project	WP		AIO		Filtering		Crops		Multi-Crops	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
Ansible	0.582	0.006	0.545	0.004	0.582	0.006	0.569	0.007	**0.649**	0.002
Brackets	0.585	0.011	0.475	0.013	0.597	0.007	0.514	0.010	**0.634**	0.006
Broadleaf	0.552	0.007	0.636	0.006	0.617	0.004	0.600	0.010	**0.651**	0.004
Camel	0.614	0.006	0.559	0.006	0.626	0.005	0.637	0.005	**0.680**	0.003
Corefx	0.389	0.008	0.486	0.012	0.390	0.008	0.437	0.010	**0.599**	0.003
Django	0.535	0.005	0.669	0.002	0.536	0.004	0.663	0.002	**0.696**	0.002
Elasticsearch	0.641	0.004	0.645	0.004	0.641	0.003	0.628	0.004	**0.706**	0.002
Fabric	0.638	0.007	0.642	0.007	0.649	0.007	0.638	0.010	**0.664**	0.006
Googleflutter	0.183	0.024	0.411	0.031	0.258	0.025	0.411	0.031	**0.577**	0.010
Homebrew	0.455	0.009	0.369	0.009	0.455	0.010	0.483	0.013	**0.577**	0.005
jGroups	0.351	0.010	0.305	0.009	0.348	0.011	0.351	0.011	**0.431**	0.007
Neutron	0.494	0.011	0.590	0.012	0.607	0.010	0.671	0.006	0.673	0.005
Node	0.586	0.005	0.623	0.003	0.648	0.007	0.621	0.003	**0.663**	0.002
Nova	0.477	0.010	0.572	0.006	0.510	0.013	0.566	0.007	**0.644**	0.006
Npm	0.520	0.007	0.608	0.007	0.521	0.007	0.608	0.007	**0.645**	0.004
Panda	0.489	0.006	0.471	0.006	0.578	0.006	0.480	0.008	**0.606**	0.004
Pytorch	0.586	0.006	0.540	0.006	0.587	0.007	0.605	0.007	**0.669**	0.002
Rails	0.481	0.006	0.483	0.006	0.508	0.007	0.483	0.006	**0.619**	0.002
Rust	0.271	0.012	0.433	0.007	0.265	0.014	0.297	0.017	**0.530**	0.008
Tensorflow	0.660	0.005	0.669	0.004	0.667	0.005	0.659	0.005	**0.690**	0.002
Tomcat	0.549	0.009	0.604	0.004	0.589	0.007	0.587	0.005	**0.637**	0.003
VScode	0.419	0.017	0.408	0.011	0.373	0.007	0.411	0.008	**0.555**	0.008
wp-Calypso	0.624	0.006	0.637	0.004	0.650	0.005	0.642	0.005	**0.679**	0.002
Ranking(RQ2)	2		1		1		1		–	
Ranking(RQ3)	3		2		2		2		1	

OOB is the base JIT-SDP model. Please refer to Table 11 for more description

during the initial time steps (0-12,000). This suggests that CroPS initially enhances Recall 1, leading to a positive impact on G-Mean. While, as time progresses, despite the project's significant imbalance, a substantial number of defect-inducing changes accumulates. In this scenario, WP achieves good performance while CroPS may introduce some uninstructive OP data due to the chosen threshold based on the initial time steps becoming less suitable over time. Furthermore, Fig. 3 also shows that the performance of the Filtering method is similar to WP, and the AIO method performs similarly to CroPS. This suggests that Filtering does not significantly contribute to selecting instructive OP data for training purposes. AIO outperforms WP during the initial time steps (0-14,000) due to the incorporation of OP defect-inducing changes, but its performance deteriorates afterwards, possibly due to the introduction of some uninstructive OP data.

Moreover, it is intriguing to delve into the change frequency of the chosen OP within CroPS. To investigate the frequency at which these selected projects undergo changes, we present two distinct measures: macro frequency (f_{ma}) and micro frequency (f_{mi}). f_{ma} is the macro frequency of a TP, and once the selected OP changes, f_{ma} will be affected. $f_{ma} = \frac{N_{ma}}{N}$,

Table 14 RQ2 & RQ3 – Average values and standard deviations of MCC on OOB

Project	WP		AIO		Filtering		Crops		Multi-Crops	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
Ansible	0.283	0.006	0.273	0.005	0.283	0.007	0.276	0.006	**0.315**	0.004
Brackets	0.293	0.009	0.240	0.010	0.306	0.008	0.263	0.009	**0.312**	0.006
Broadleaf	0.230	0.007	**0.315**	0.008	0.268	0.006	0.288	0.010	0.309	0.009
Camel	0.323	0.005	0.292	0.005	0.327	0.006	0.343	0.006	**0.375**	0.004
Corefx	0.220	0.006	0.251	0.012	0.220	0.006	0.246	0.006	**0.312**	0.004
Django	0.300	0.005	0.407	0.004	0.302	0.004	0.400	0.003	**0.413**	0.003
Elasticsearch	0.374	0.003	0.374	0.005	0.374	0.003	0.394	0.004	**0.429**	0.003
Fabric	0.328	0.011	0.321	0.012	0.336	0.012	0.323	0.015	0.336	0.011
Googleflutter	0.094	0.017	0.137	0.037	0.149	0.020	0.137	0.037	**0.210**	0.015
Homebrew	0.260	0.006	0.225	0.006	0.261	0.007	0.279	0.009	**0.338**	0.004
jGroups	0.166	0.007	0.150	0.007	0.165	0.009	0.166	0.008	**0.192**	0.006
Neutron	0.173	0.010	0.309	0.011	0.281	0.012	**0.369**	0.008	0.357	0.009
Node	0.289	0.007	0.298	0.004	0.334	0.008	0.308	0.004	**0.339**	0.003
Nova	0.208	0.007	0.292	0.007	0.229	0.010	0.289	0.005	**0.318**	0.008
Npm	0.198	0.007	0.264	0.008	0.198	0.007	0.264	0.008	**0.311**	0.008
Panda	0.253	0.005	0.237	0.005	**0.306**	0.006	0.226	0.008	0.292	0.006
Pytorch	0.321	0.005	0.304	0.005	0.304	0.005	0.338	0.007	**0.383**	0.003
Rails	0.244	0.004	0.247	0.004	0.259	0.005	0.247	0.004	**0.320**	0.003
Rust	0.157	0.010	0.203	0.007	0.146	0.011	0.179	0.013	**0.263**	0.010
Tensorflow	0.373	0.005	0.365	0.005	0.380	0.006	0.371	0.004	**0.393**	0.003
Tomcat	0.226	0.011	0.267	0.005	0.250	0.008	0.289	0.005	**0.295**	0.006
VScode	0.256	0.011	0.232	0.009	0.229	0.005	0.255	0.008	**0.329**	0.007
wp-Calypso	0.349	0.005	0.351	0.004	0.355	0.006	0.354	0.004	**0.373**	0.004
Ranking(RQ2)	4		3		2		1		–	
Ranking(RQ3)	5		4		3		2		1	

OOB is the base JIT-SDP model. Please refer to Table 11 for more description

where N is the number of similarity updates, N_{ma} is the number of selected OP changes. f_{mi} is the micro frequency of a OP on a TP, and once the OP is selected or discarded, f_{mi} will be affected. $f_{mi} = \frac{N_{mi}}{N}$, where N is the number of similarity updates, N_{mi} corresponds to the number of times the OP is selected or discarded.

Table 17 shows the macro frequency of each projects when base model is ODaSC, and Table 18 shows the micro frequency on Fabric, which has the highest macro frequency. The θ_{crops} in Table 17 is the selection threshold used for CroPS. As can be seen, a threshold that is too large or too small will reduce the change frequency of the selected OP. When the threshold is too large, OP will be more difficult to be selected, as shown in Fig. 4. When the threshold is too small, OP will be more easy to be selected, as shown in Fig. 5. When the threshold is moderate, the selected OP will change frequently, as shown in Fig. 6.

Therefore, the change frequency of selected projects is related to the threshold of CroPS, which confirms the importance of threshold selection in CroPS.

Table 15 RQ2 & RQ3 – average values and standard deviations of G-mean on PBSA

Project	WP		AIO		Filtering		Crops		Multi-Crops	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
Ansible	0.593	0.005	0.612	0.004	0.606	0.004	0.618	0.003	**0.649**	0.003
Brackets	0.627	0.008	0.613	0.008	0.623	0.010	0.630	0.011	**0.637**	0.009
Broadleaf	0.545	0.017	0.636	0.008	0.592	0.011	0.596	0.010	**0.642**	0.007
Camel	0.652	0.011	0.651	0.008	0.665	0.009	0.637	0.009	**0.671**	0.003
Corefx	0.574	0.009	0.628	0.007	0.571	0.010	0.630	0.007	**0.633**	0.006
Django	0.560	0.005	0.695	0.001	0.689	0.003	0.695	0.002	0.694	0.001
Elasticsearch	0.655	0.002	0.692	0.002	0.655	0.002	0.693	0.002	**0.704**	0.001
Fabric	0.628	0.008	0.645	0.009	0.645	0.007	0.644	0.010	**0.661**	0.005
Googleflutter	0.103	0.012	0.386	0.019	0.185	0.024	0.386	0.019	**0.561**	0.022
Homebrew	0.509	0.005	0.360	0.006	0.510	0.009	0.508	0.012	**0.610**	0.006
jGroups	0.528	0.003	0.465	0.005	0.521	0.007	0.517	0.005	**0.553**	0.004
Neutron	0.503	0.012	0.693	0.004	0.619	0.011	0.690	0.007	0.685	0.007
Node	0.596	0.006	0.634	0.004	0.627	0.007	0.595	0.005	**0.664**	0.002
Nova	0.488	0.008	0.654	0.003	0.506	0.013	0.653	0.004	**0.657**	0.003
Npm	0.523	0.015	0.608	0.014	0.544	0.014	0.612	0.011	**0.625**	0.009
Panda	0.542	0.022	**0.646**	0.005	0.636	0.009	0.632	0.008	0.636	0.006
Pytorch	0.640	0.003	**0.678**	0.002	0.646	0.006	0.642	0.005	0.670	0.002
Rails	0.551	0.008	0.581	0.006	0.551	0.008	0.552	0.006	**0.652**	0.002
Rust	0.383	0.014	0.508	0.010	0.359	0.011	0.395	0.013	**0.595**	0.006
Tensorflow	0.657	0.009	0.659	0.005	0.653	0.005	0.628	0.004	**0.667**	0.004
Tomcat	0.546	0.015	0.606	0.008	0.523	0.014	0.594	0.008	**0.627**	0.002
VScode	0.602	0.012	0.544	0.017	0.605	0.012	0.554	0.016	**0.687**	0.007
wp-Calypso	0.635	0.007	0.644	0.004	0.659	0.005	0.645	0.003	**0.677**	0.002
Ranking(RQ2)	4		1		3		2		–	
Ranking(RQ3)	5		2		4		3		1	

PBSA is the base JIT-SDP model. Please refer to Table 11 for more description

Answer to RQ2: We proposed CroPS based on the project-level similarity metrics designed in RQ1 to select instructive other projects over time for online CP JIT-SDP. Experimental results demonstrated the significant effectiveness of the proposed CroPS compared to WP and other online CP methods, showing it as a promising online CP method. When compared to other online CP methods, CroPS could perform either significantly better than or on par with the AIO and Filtering methods, showing the competitiveness of CroPS among state-of-the-art online CP methods.

6.3 RQ3: Predictive Performance of Multi-CroPS

This section aims to investigate predictive performance of Multi-CroPS, an enhanced version of CroPS, completing our answer to RQ3. Experimental results are presented in Tables 11, 12, 13, 14, 15, and 16 in terms of G-Mean and MCC using different base methods in Section 5.1,

Table 16 RQ2 & RQ3 – average values and standard deviations of MCC on PBSA

Project	WP		AIO		Filtering		Crops		Multi-Crops	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
Ansible	0.279	0.007	0.313	0.004	0.290	0.005	**0.322**	0.004	0.318	0.005
Brackets	0.304	0.011	0.294	0.007	0.286	0.016	0.310	0.013	0.299	0.011
Broadleaf	0.230	0.013	**0.302**	0.012	0.240	0.010	0.273	0.013	0.292	0.013
Camel	0.338	0.013	0.335	0.010	0.352	0.012	0.328	0.009	0.351	0.007
Corefx	0.298	0.008	0.271	0.012	0.300	0.011	0.274	0.012	0.277	0.012
Django	0.321	0.005	0.421	0.002	0.414	0.004	**0.425**	0.002	0.405	0.002
Elasticsearch	0.384	0.002	0.419	0.004	0.384	0.002	0.420	0.003	**0.428**	0.002
Fabric	0.314	0.011	0.321	0.013	0.325	0.012	0.327	0.015	0.332	0.011
Googleflutter	0.057	0.008	0.124	0.026	0.103	0.019	0.124	0.026	**0.200**	0.031
Homebrew	0.301	0.005	0.218	0.006	0.302	0.008	0.308	0.009	**0.367**	0.006
jGroups	0.200	0.005	0.219	0.004	0.214	0.008	0.205	0.006	0.210	0.005
Neutron	0.174	0.009	0.397	0.008	0.270	0.015	**0.398**	0.013	0.383	0.013
Node	0.300	0.005	0.308	0.004	0.323	0.006	0.302	0.005	**0.337**	0.004
Nova	0.211	0.008	0.334	0.004	0.224	0.010	**0.336**	0.006	0.332	0.005
Npm	0.179	0.017	0.248	0.020	0.197	0.019	0.268	0.013	**0.273**	0.013
Panda	0.269	0.014	0.323	0.008	0.320	0.010	0.296	0.012	0.299	0.012
Pytorch	0.343	0.005	**0.383**	0.003	0.349	0.007	0.346	0.006	0.364	0.003
Rails	0.273	0.007	0.301	0.005	0.274	0.006	0.275	0.004	**0.335**	0.003
Rust	0.216	0.010	0.233	0.012	0.189	0.009	0.217	0.012	**0.295**	0.007
Tensorflow	0.362	0.009	0.342	0.007	0.357	0.005	0.336	0.004	0.358	0.005
Tomcat	0.218	0.013	0.254	0.012	0.210	0.015	**0.284**	0.007	0.268	0.005
VScode	0.378	0.012	0.281	0.018	0.380	0.013	0.332	0.014	**0.399**	0.012
wp-Calypso	0.358	0.006	0.356	0.005	0.363	0.006	0.356	0.004	**0.372**	0.004
Ranking(RQ2)	3		1		2		1		–	
Ranking(RQ3)	4		2		3		2		1	

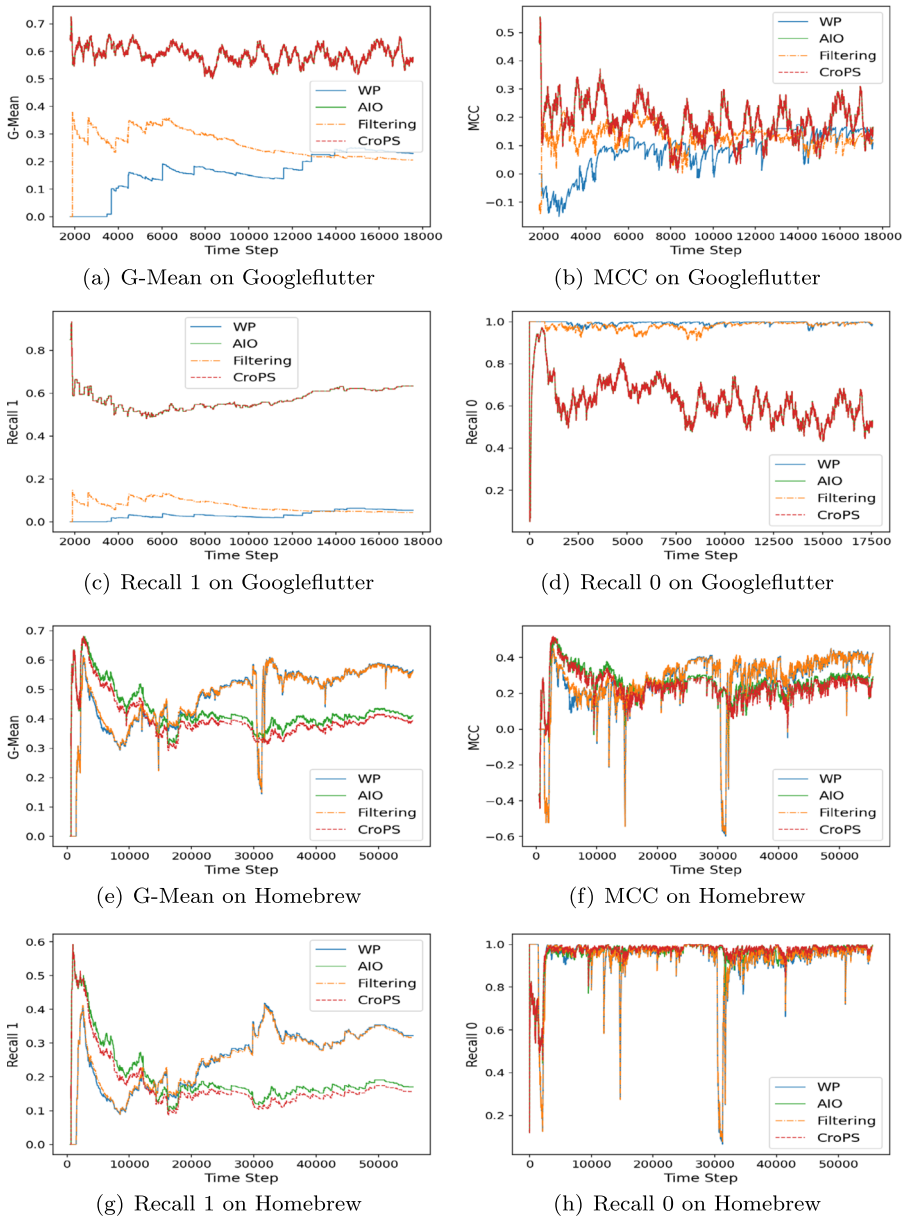
PBSA is the base JIT-SDP model. Please refer to Table 11 for more description

individually. The row labeled “Ranking(RQ3)" represents the ranking of the competing methods, determined using the double Scott-Knott ESD test.

As we can see from those tables, the experimental results consistently demonstrate that Multi-CroPS outperforms other methods, including WP, AIO, Filtering, and the proposed CroPS, in terms of both G-Mean and MCC using all base JIT-SDP models. These results indicate that Multi-CroPS exhibits significant advantages over state-of-the-art online CP methods including the proposed CroPS, demonstrating the effectiveness of Multi-CroPS in identifying instructive OP data.

To gain a deeper understanding of the advantage of the advanced Multi-CroPS, we choose to analyse the projects that exhibit the best and worst performance improvement ratios in terms of G-Mean when comparing CroPS to Multi-CroPS. These analyses would be conducted using the base method ODaSC for it usually getting the best predictive performance against other base JIT-SDP models with respect to different datasets and in terms of different performance metrics.

As shown in Table 11, Rust exhibits the best performance improvement ratio from CroPS to Multi-CroPS, with an improvement ratio of 54.9%. Figure 7 provides insights into the online



(a) G-Mean on Googleflutter (b) MCC on Googleflutter

(c) Recall 1 on Googleflutter (d) Recall 0 on Googleflutter

(e) G-Mean on Homebrew (f) MCC on Homebrew

(g) Recall 1 on Homebrew (h) Recall 0 on Homebrew

Fig. 3 RQ2 – online predictive performance of the WP (ODaSC) and CP methods on two representative datasets including Googleflutter and Homebrew. CroPS has the highest performance improvement ratio compared to the WP method on Googleflutter; it has the worse performance improvement ratio compared to WP on Homebrew

predictive performance of different CP methods throughout the test steps on the Rust project. WP and Filtering yield similar G-Mean values, while CroPS slightly outperforms them in terms of G-Mean. AIO and Multi-CroPS gain higher G-Mean values, with Multi-CroPS consistently outperforming AIO beyond time step 40,000. In other words, Multi-CroPS can get the best overall G-Mean performance on this dataset. According to Table 3, Rust is a highly imbalanced project, with a proportion of defect-inducing changes of only 2.03%. WP,

Table 17 The macro change frequency of each project when base model is ODaSC

Project	N	N_{ma}	f_{ma}	θ_{crops}
Ansible	212	131	61.79%	0.2
Brackets	23	18	78.26%	0.25
Broadleaf	133	90	67.67%	0.3
Camel	88	16	18.18%	0.5
Corefx	262	90	34.35%	0.3
Django	217	106	48.85%	0.15
Elasticsearch	227	37	16.30%	0.5
Fabric	123	105	85.37%	0.3
Googleflutter	176	0	0.00%	0.1
Homebrew	112	49	43.75%	0.15
jGroups	94	3	3.19%	0.5
Neutron	17	12	70.59%	0.3
Node	275	26	9.45%	0.5
Nova	229	169	73.80%	0.25
Npm	76	2	2.63%	0.15
Panda	44	12	27.27%	0.3
Pytorch	136	6	4.41%	0.5
Rails	165	103	62.42%	0.2
Rust	684	311	45.47%	0.25
Tensorflow	129	65	50.39%	0.3
Tomcat	62	10	16.13%	0.5
VScode	103	0	0.00%	0.1
wp-Calypso	344	65	18.90%	0.15

N is the number of similarity updates. N_{ma} is the number of selected OP changes. f_{ma} is the macro change frequency, and θ_{crops} is the selection threshold used for CroPS

Filtering, and CroPS exhibit exceptionally high Recall 0, indicating their ability in deciding clean software changes. However, their performance in terms of Recall 1 is considerably low, suggesting their lack of ability in detecting defect-inducing changes. Specially, after time step 30,000, Recall 1 of the WP, Filtering, and CroPS methods all drop below 0.2, showing their inability in detecting defect-inducing changes effectively. Among them, CroPS incorporates instructive OP data and leads to an improvement in Recall 1, though the improvement is not significant. AIO further demonstrates significantly better Recall 1 by incorporating all OP data to supplement defect-inducing changes. Multi-CroPS consists of different CroPS models, each capable of incorporating OP data at supplementing defect-inducing changes at different scales. The design of Multi-CroPS is intentionally focused on effectively capturing defect-inducing changes, thereby exhibiting a specific bias towards the prediction on the defect-inducing class, which in return, contributes to improved Recall 1 and ultimately improved overall performance in terms of G-Mean and MCC.

As shown in Table 11, Elasticsearch exhibits the worst performance improvement from CroPS to Multi-CroPS, with a decrease of 0.9%, showing an example (Elasticsearch) that Multi-CroPS may have a negative impact on CroPS. Figure 7 provides insights into the online predictive performance of different CP methods throughout the test steps on Elasticsearch. WP, Filtering, CroPS, and Multi-CroPS demonstrate competitive G-Mean values, whereas AIO exhibits the poorest performance in G-Mean. Elasticsearch is relatively balanced (see

Table 18 The micro change frequency of Fabric when base model is ODaSC

Project	N_{mi}	f_{mi}
Ansible	0	0.00%
Brackets	1	0.81%
Broadleaf	49	39.84%
Camel	41	33.33%
Corefx	1	0.81%
Django	0	0.00%
Elasticsearch	53	43.09%
Fabric	0	0.00%
Googleflutter	1	0.81%
Homebrew	0	0.00%
jGroups	37	30.08%
Neutron	49	39.84%
Node	27	21.95%
Nova	13	10.57%
Npm	49	39.84%
Panda	0	0.00%
Pytorch	5	4.07%
Rails	0	0.00%
Rust	1	0.81%
Tensorflow	7	5.69%
Tomcat	41	33.33%
VScode	0	0.00%
wp-Calypso	1	0.81%

N is 123 which can be seen in Table 17. N_{mi} corresponds to the number of times Fabric is selected or discarded. f_{mi} is the micro change frequency

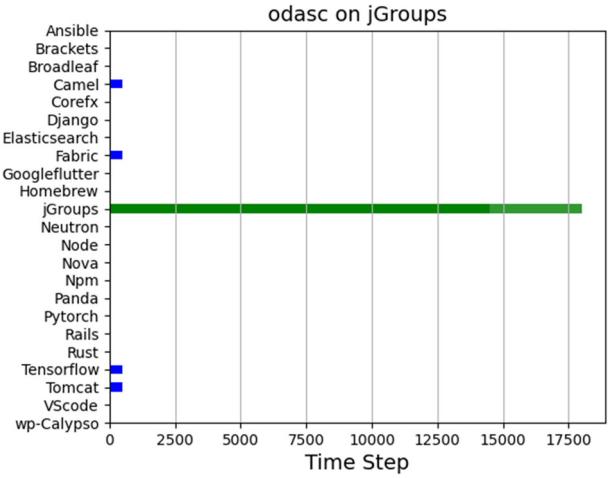


Fig. 4 The selected projects on jGroups with θ_{crops} as 0.5. The blue bars indicate that projects were selected to participate in model training during the corresponding time steps. The green bars related to the TP. In this case, only four OP are selected in the initial phase, only TP will participate in model updating in the following time steps

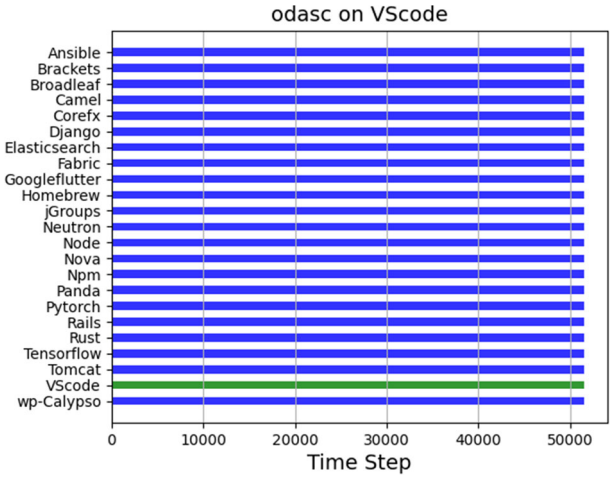


Fig. 5 The selected projects on VScode with θ_{crops} as 0.1. The blue bars indicate that projects were selected to participate in model training during the corresponding time steps. The green bars related to the TP. In this case, all projects are selected, and there is no change throughout

Table 3), meaning it does not suffer from a severe shortage of defect-inducing changes. In this case, WP can maintain a relatively good Recall 1, being above 0.6 after time step 10,000. On the other side, AIO may incorporate uninstructive OP data during model updating, which may cause the model to become biased towards defect-inducing changes. Even this operation may enable AIO to achieve higher Recall 1 compared to other methods, it may also deteriorate the prediction power when it comes to clean changes, resulting in low Recall 0. Multi-CroPS can achieve a relatively balanced performance on both defect-inducing changes and clean changes, and thus CroPS and Multi-CroPS demonstrate similar predictive performance.

In addition, we provide a new perspective to analyze the impact of Multi-CroPS compared to CroPS. Multi-CroPS has an “ensemble" mechanism to try to “predict label 1". This predicted “1" can be either better or worse than CroPS do. Table 19 shows the effect of

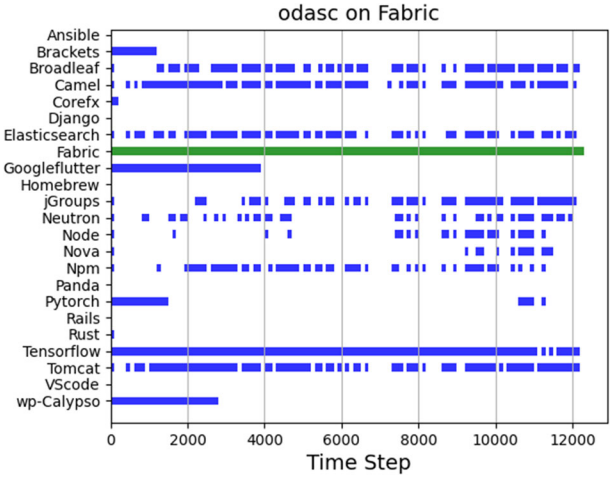
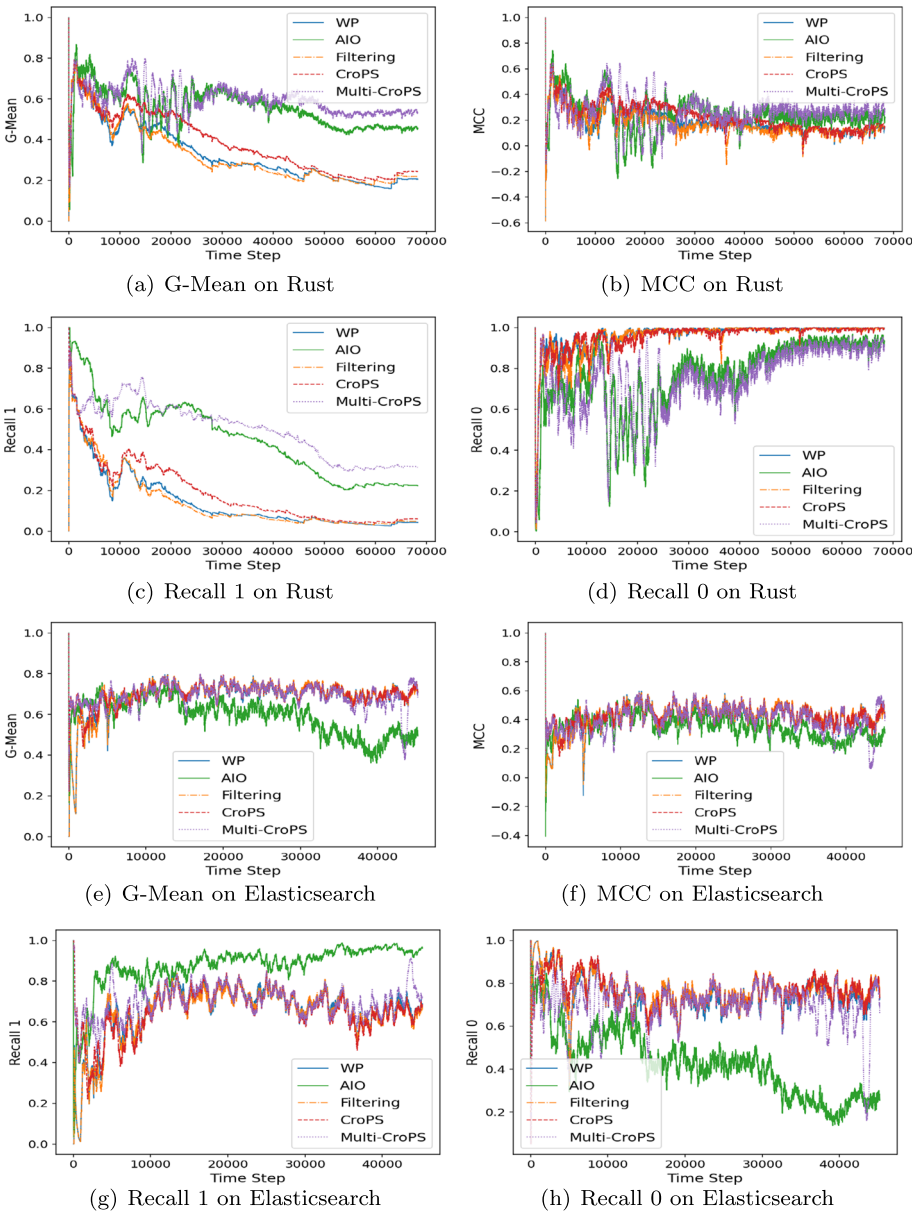


Fig. 6 The selected projects on fabric with θ_{crops} as 0.3. The blue bars indicate that projects were selected to participate in model training during the corresponding time steps. The green bars related to the TP. In this case, the selected OP changes very frequently



(a) G-Mean on Rust (b) MCC on Rust

(c) Recall 1 on Rust (d) Recall 0 on Rust

(e) G-Mean on Elasticsearch (f) MCC on Elasticsearch

(g) Recall 1 on Elasticsearch (h) Recall 0 on Elasticsearch

Fig. 7 RQ3 – online performance of the WP (ODaSC) and CP methods on rust and Elasticsearch. Multi-CroPS has the highest performance improvement ratio compared to CroPS on rust and has the worse performance improvement ratio compared to CroPS on Elasticsearch

"ensemble" mechanism in Multi-CroPS on CroPS based on ODaSC. In the table, "TP/defect" is the number of changes in Multi-CroPS that make the prediction better as a percentage of all defect changes, that is, the positive effects of Multi-CroPS on defect changes. "FP/clean" is the number of changes in Multi-CroPS that make the prediction worse as a percentage of all clean changes, that is, the negative effects of Multi-CroPS on clean changes. We conduct

Table 19 Effect of “ensemble" mechanism in Multi-CroPS on CroPS based on ODaSC

Project	P	tp	fp	Precision	1-Precision	tp/defect	fp/clean
Ansible	1626	665	961	40.90%	59.10%	4.40%	3.50%
Brackets	0	0	0	–	–	0.00%	0.00%
Broadleaf	8	2	6	20.70%	79.30%	0.10%	0.10%
Camel	2692	553	2140	20.50%	79.50%	8.90%	9.00%
Corefx	7434	653	6782	8.80%	91.20%	36.10%	27.80%
Django	182	65	117	35.80%	64.20%	0.60%	0.80%
Elasticsearch	2490	728	1762	29.20%	70.80%	4.60%	6.00%
Fabric	0	0	0	–	–	0.00%	0.00%
Googleflutter	978	15	963	1.60%	98.40%	6.80%	5.60%
Homebrew	4697	214	4483	4.60%	95.40%	23.10%	10.50%
jGroups	2224	361	1863	16.20%	83.80%	11.50%	12.60%
Neutron	0	0	0	–	–	0.00%	0.00%
Node	3600	1260	2340	35.00%	65.00%	14.30%	13.30%
Nova	67	38	30	55.90%	44.10%	0.40%	0.20%
Npm	80	17	63	21.60%	78.40%	1.30%	1.00%
Panda	0	0	0	–	–	0.00%	0.00%
Pytorch	78	28	50	35.40%	64.60%	0.30%	0.30%
Rails	7506	2601	4905	34.70%	65.30%	18.30%	11.70%
Rust	14127	387	13740	2.70%	97.30%	27.90%	20.50%
Tensorflow	930	156	774	16.80%	83.20%	1.00%	1.60%
Tomcat	1619	491	1128	30.30%	69.70%	9.50%	8.40%
VScode	2891	139	2753	4.80%	95.20%	11.90%	5.50%
wp-Calypso	233	31	202	13.30%	86.70%	0.50%	0.90%

Here shows the average results on 20 runs. “P" is the number of code changes in which Multi-CroPS affects, that is, the Multi-CroPS gives a “1" label while CroPS gives a “0" label. “tp" is the number of defect changes which is affected by Multi-CroPS. For these changes, Multi-CroPS makes the prediction better. “fp" is the number of clean changes which is affected by Multi-CroPS. For these changes, Multi-CroPS makes the prediction worse. “tp/defect" is the number of changes in Multi-CroPS that make the prediction better as a percentage of all defect changes, that is, the positive effects of Multi-CroPS on defect changes. “fp/clean" is the number of changes in Multi-CroPS that make the prediction worse as a percentage of all clean changes, that is, the negative effects of Multi-CroPS on clean changes

a one-sided Wilcoxon signed-rank test to compare the “TP/defect" and “FP/clean" across all projects. The p-value is 0.0383, which means that the positive effects of Multi-CroPS on defect changes are significantly better than the negative effects on clean changes. Therefore, the “ensemble" mechanism of Multi-CroPS has a significant positive effect to CroPS.

6.3.1 Higher Computational Cost of Multi-CroPS

The trade-off for the superior predictive performance of Multi-CroPS is its relatively higher computational cost compared to WP and other CP methods including the proposed CroPS. Average time costs, measured in seconds, for completing one commit (software change), including both the training and prediction processes, on an Intel(R) Xeon(R) Gold 623BR CPU at 2.20GHz and 128Gb of RAM, are summarized in Table 20.

We can see from Table 20 that Multi-CroPS incurs higher computational costs compared to other methods, primarily due to the maintenance of multiple CroPS models simultane-

Table 20 Average time costs for one commit, measured in seconds

Method	WP	AIO	Filtering	CroPS	Multi-CroPS
Time (s)	0.0432	0.6398	0.3370	0.1449	1.4975

ously. The relationship between time costs and the number of CroPS models in Multi-CroPS is not a simple linear one. This is because each CroPS model in Multi-CroPS has a different threshold, leading to the selection of varying amounts of OP data for model updates. Consequently, Multi-CroPS requires more computational resources and exhibits significantly larger time costs compared to CroPS. It is noteworthy that despite Multi-CroPS having an average running time of 1.4975 seconds per commit (software change), this duration includes both the training and prediction processes. Thus, the time cost for predicting each software change could be much less than 1.4975 seconds. This amount of time costs is generally acceptable for developers as it enables timely predictions and decision-making during software development and maintenance processes. In addition, the training process of Multi-CroPS can be parallelized by utilizing a separate processor for each base CroPS using one processor, further reducing the running time per commit.

We can also see from Table 20 that CroPS demonstrates lower time costs in comparison to the AIO and Filtering methods. AIO requires updating the model with every incoming training example produced from both OP and TP data, resulting in higher time costs. On the other side, Filtering needs to check each OP data to determine its relevance for model updating, which further increases processing time. In contrast, CroPS primarily incurs time costs in calculating the project-level similarities, which is performed at regular time intervals (operated with the hyper-parameter t_{up}). The selection threshold in CroPS also contributes to time consumption, as a smaller threshold incorporates more OP data for training purposes. However, even with a smaller threshold, the amount of data used for model updating in CroPS does not exceed that of AIO, probably contributing to lower time costs.

From a practical perspective, Multi-CroPS offers superior predictive performance at the expense of higher computational costs compared to CroPS and other online CP methods. Therefore, we recommend adopting Multi-CroPS when sufficient computing resources are available and the timely response is not of an extreme demand. However, in cases when computing resources are limited or an extremely prompt response is required, CroPS is recommended. CroPS can maintain competitive predictive performance while requiring lower computational costs than Multi-CroPS.

Answer to RQ3: We proposed Multi-CroPS as an improved version of CroPS, which incorporates multiple scales of OP selections in CroPS and a defect-inducing prediction biased mechanism into the learning framework. Experimental results demonstrated the significant superiority of Multi-CroPS over WP and other online CP methods including the proposed CroPS. Compared to other methods, Multi-CroPS show better G-Mean and MCC on all base JIT-SDP models, which highlights the strong competitiveness of Multi-CroPS. However, this improved performance comes at the expense of higher computational costs compared to CroPS, primarily due to the maintenance of multiple CroPS models. Therefore, we recommend adopting Multi-CroPS when sufficient computing resources are available and the timely response is not of an extreme demand. Conversely, in situations where computing resources are limited or an extremely prompt response is required, we recommend using CroPS.

7 Discussion

7.1 Analysis of Diversity and Validity

To enhance the comprehensiveness and credibility of our experimental findings, we undertook a quantitative analysis of the diversity and validity of the CP data incorporated by various CP methods. Notably, these analyses were also performed using the base method ODaSC, which usually getting the best predictive performance against other base JIT-SDP models.

As expounded upon in Section 1, the concept of "data diversity" is related to data distribution, referring to the underlying fact that data from OP does not closely resemble the data from the TP. Broader data distribution in OP usually indicates higher data diversity. The concept of "data validity", on the other hand, is related to data similarity, referring to the underlying fact that the OP data selected for training the model should be effective in improving the predictive performance of JIT-SDP. These two concepts are formulated for providing a deeper understanding of CP methods and will be formulated based on the variance and distance metrics.

Data diversity is defined as

$$Div = 1 - \frac{1}{1 + \frac{1}{d}\sum_{i}^{d} \sigma_i} \tag{4}$$

where σ_i denotes the variance of the feature values in the ith dimension of the CP data and d denotes the total number of data dimensions. The CP data includes all the data from the Other Projects (OPs) and the Target Project (TP). It is worth noting that there are multiple CroPS models in Multi-CroPS, so we calculate the data diversity for each submodel, and use the average diversity as the data diversity of Multi-CroPS.

Data validity is defined as

$$Val = 1 - \frac{1}{2}(Dis_f + Dis_c) \tag{5}$$

where Dis_f denotes a fine-grained distance and Dis_c denotes a coarse-grained distance. Specifically, the fine-grained distance is formulated as

$$Dis_f = \frac{1}{n}\sum_{j}^{n} dis_j,$$

following Tabassum et al. (2022) in calculating their data-level distance between pairs of data points, where dis_j denotes the minimum distance between the j-th CP data point and the data points of same class in the WP, and n denotes the total number of CP data. In this way, the fine-grained distance captures the similarity between the CP data and the WP data at the individual data point level. It is pertinent to mention that during the calculation of the fine-grained distance, a noteworthy consideration is the exclusion of TP data from the CP data set. This exclusion is based on the understanding that TP data inherently possesses a distance of 0 to TP data itself. By excluding TP data from the calculation, we ensure the accuracy and integrity of the fine-grained distance measurement. The coarse-grained distance (Dis_c) is formulated as the distance of the Spearman Correlation of the whole TP and CP data points, inspired by Kamei et al. (2016) in calculating their project-level distance. In this way, the coarse-grained distance provides a higher-level, aggregated measure of the distance between the overall CP and WP data distributions.

In order to make the data validity more objective and reasonable, the min-max normalization (across all projects and CP methods) is used to the fine-grained distance and the coarse-grained distance respectively. This normalization ensures that fine-grained distance and coarse-grained distance have an equal impact on determining data validity. Consequently, both the fine-grained distance (Dis_f) and the coarse-grained distance (Dis_c) are scaled to fall within the range of [0, 1]. The data diversity metric in (4) integrates both the fine-grained distance Dis_f and the coarse-grained distance Dis_c, offering a more comprehensive and robust assessment of the data validity. For Multi-CroPS, the distances are calculated individually for each submodel, and the average distance is subsequently used to determine the data validity of Multi-CroPS. It is worth noting that the defined validity Val is an approximation of the intrinsic "validity" in practical settings, as its calculation process requires the true labels of all data points, which cannot be readily accessible in reality due to verification latency. In this paper, we deliberately utilize true labels to calculate the true validity and investigate the trade-off between validity and diversity of the CP methods.

In the prediction process, the old data becomes obsolete, and the model focuses more on recent data. Therefore, in order to more accurately evaluate the diversity and validity, we

Table 21 Average diversity of CP methods across time steps

Project	AIO	Filtering	CroPS	Multi-CroPS
Ansible	0.843	0.811	0.831	0.837
Brackets	0.828	0.660	0.824	0.804
Broadleaf	0.823	0.699	0.807	0.809
Camel	0.817	0.723	0.804	0.814
Corefx	0.844	0.734	0.822	0.826
Django	0.805	0.703	0.804	0.774
Elasticsearch	0.841	0.800	0.806	0.831
Fabric	0.829	0.681	0.811	0.817
Googleflutter	0.849	0.711	0.849	0.823
Homebrew	0.851	0.826	0.841	0.837
jGroups	0.757	0.672	0.702	0.754
Neutron	0.832	0.722	0.824	0.815
Node	0.836	0.813	0.835	0.831
Nova	0.823	0.710	0.803	0.803
Npm	0.821	0.716	0.821	0.815
Panda	0.839	0.743	0.826	0.821
Pytorch	0.853	0.755	0.814	0.840
Rails	0.807	0.749	0.809	0.796
Rust	0.838	0.760	0.818	0.821
Tensorflow	0.851	0.756	0.838	0.839
Tomcat	0.813	0.671	0.794	0.808
VScode	0.848	0.658	0.848	0.814
wp-Calypso	0.846	0.695	0.846	0.815
Average	0.830	0.729	0.816	0.815
Median	0.836	0.722	0.821	0.815
Rank	1	4	2	2

ODaSC is the base JIT-SDP model. The ranking of the method is calculated using Scott-Knott ESD test

used a sliding window to maintain recent data for each project. We set the sliding window size to 500. It is the minimum candidate value of sliding window size in Filtering and CroPS, which has strong statistical significance and small computational cost.

Table 21 shows the average diversity across all time steps for CP methods. We can see that AIO shows the strongest diversity, while Filtering shows the weakest. AIO incorporates all available OP data into the model training process and has the strongest diversity. The OP data incorporated by Filtering is limited to a certain range (near TP data), and its diversity is relatively weak. CroPS and Multi-CroPS selectively incorporate data from instructive OP, and the data distribution ranges between AIO and Filtering, so is their diversity.

Table 22 shows the average validity across all time steps for CP methods. We can see that Filtering and Multi-CroPS show the relatively strong validity, while AIO and CroPS show the relatively weak. Filtering only incorporates OP data that is similar to TP data, which has relatively strong validity. AIO may incorporates data that is not related to TP, which has relatively weak validity. CroPS incorporates data from instructive OP. However, it is important to acknowledge that there may exist certain data points that are not directly relevant to TP data in instructive OP. Consequently, the validity of CroPS is relatively weaker

Table 22 Average validity of CP methods across time steps

Project	AIO	Filtering	CroPS	Multi-CroPS
Ansible	0.385	0.575	0.360	0.449
Brackets	0.540	0.520	0.459	0.570
Broadleaf	0.491	0.537	0.544	0.528
Camel	0.467	0.338	0.557	0.527
Corefx	0.389	0.485	0.468	0.437
Django	0.333	0.458	0.423	0.588
Elasticsearch	0.345	0.581	0.464	0.411
Fabric	0.431	0.501	0.506	0.491
Googleflutter	0.411	0.510	0.399	0.507
Homebrew	0.127	0.967	0.067	0.179
jGroups	0.538	0.853	1.000	0.678
Neutron	0.410	0.396	0.374	0.452
Node	0.486	0.491	0.270	0.435
Nova	0.448	0.489	0.400	0.486
Npm	0.218	0.447	0.319	0.337
Panda	0.388	0.457	0.433	0.470
Pytorch	0.447	0.427	0.392	0.438
Rails	0.549	0.432	0.509	0.682
Rust	0.416	0.536	0.379	0.485
Tensorflow	0.354	0.360	0.483	0.418
Tomcat	0.536	0.452	0.407	0.498
VScode	0.296	0.617	0.329	0.466
wp-Calypso	0.503	0.360	0.474	0.591
Average	0.413	0.513	0.435	0.484
Median	0.416	0.489	0.423	0.485
Rank	2	1	2	1

ODaSC is the base JIT-SDP model. The ranking of the method is calculated using Scott-Knott ESD test

compared to the Filtering method, which specifically excludes OP data that is unrelated to TP. Nonetheless, the exclusion of OP that is unrelated to TP in CroPS contributes to a slightly stronger validity compared to AIO. This observation is supported by the average and median values presented in Table 22. Regarding Multi-CroPS, its consideration of CroPS models under different thresholds renders it more adaptable to the current data environment. This adaptability enhances its suitability and contributes to its generally higher validity compared to the CroPS model with a single threshold.

In overall assessment, it can be observed that AIO exhibits the strongest diversity but the weakest validity. Conversely, Filtering demonstrates the strongest validity but the weakest diversity. The diversity and validity of both CroPS and Multi-CroPS fall within the spectrum between these two extremes.

7.2 Conjecture About Zigzags

It is worth noting that there are significant zigzag in Figs. 3 and 7. This is a common phenomenon caused by Concept Drift (CD) in online JIT-SDP (Tabassum et al. 2022; Cabral et al. 2023; Cabral and Minku 2022; Song et al. 2022). In the early stages of the project, the magnitude of zigzag is generally large. This may be due to the fact that the model is in its infancy and the understanding of Target Project (TP) data is not stable. In the subsequent prediction process, there are frequent zigzag. The CD of high frequency and large amplitude exists in online JIT-SDP, which causes frequent zigzag.

From Figs. 3 and 7, we can see that the zigzag with the largest amplitude usually appears on Recall0, which indicates that Recall0 is most affected by CD in these projects, especially when using the AIO method. AIO incorporates all external data to the model update process. Considering that the amount of clean change is usually more than the amount of defect change, the amount of clean change incorporated by AIO is usually larger. When concept drift occurs, because the model has learned a lot of data under the old concept, it is difficult to adapt to the new concept, and its performance will be drastically reduced. However, when a large number of training data under the new concept is reached, the model can quickly adapt to the new concept, and the predictive performance quickly recovers. Such a process may be the cause of zigzag, and the larger the amount of data used in training, the larger the amplitude of zigzag might be. This explains the larger amplitude of zigzag occurs in Recall0 in this paper.

Therefore, we suspect that the amplitude and frequency of the zigzag are related to CD, and the amplitude is also related to the amount of training data. When developers need a more stable model, we recommend appropriately reducing the amount of data added by CP approaches, such as increasing the selection threshold of CroPS.

8 Threats to Validity

Internal Validity. To mitigate threats related to the randomness of online JIT-SDP model, we conducted experiments 20 times for each WP and CP method on each projects. To ensure a fair comparison, we employed grid search for parameter tuning, which was done using an initial portion of the data stream from each project. There may be some defect-inducing changes suffering from extremely large verification latency, being unable to collect their true labels at the time of data production. Consequently, there is non-zero possibility of misclassifying them as clean changes. To mitigate the related threats, we excluded software changes from the

later periods of the data streams, specifically the last six months following previous setting up Kamei et al. (2013). Similar to previous work that has adopted the same datasets (Cabral et al. 2019; Song et al. 2022), other threats to internal validity include various of noise arising from the SZZ algorithm. To mitigate the related threats, we conducted manual Kappa analysis in Section 4. We also adopted the waiting time of 15 days that has been shown to lead to a good trade-off between the label noise resulting from verification latency and the obsolescence of the trained models (Song et al. 2022).

External Validity. We have investigated 23 open-source projects, covering a wide range of data characteristics as explained in Section 4. Some of these projects were previously used and published in similar studies (Song et al. 2023b, a; Cabral et al. 2019). To enable potential comparisons with those previous studies and reuse existing data resources, we utilized those publicly available projects as part of our datasets. However, as with any study involving machine learning, experimental conclusions may not generalize well to other projects or other learning machines. Including more projects is worthwhile for future experimental studies. We investigated the proposed online CP methods based on popular online JIT-SDP methods, including OOB, ODaSC, and PBSA. Other online learning machines may get different results and need further investigations.

Construct Validity. This study primarily focuses on G-Mean and MCC as evaluation metrics. G-Mean is known for its robustness to class imbalance (Wang et al. 2018; Cabral et al. 2019), making it well-suited to JIT-SDP that often suffers from the class imbalance issue. MCC is adopted as it has become popular in the area of JIT-SDP (Chicco and Jurman 2020; Chicco et al. 2021), which uses all entries of the confusion matrix. To track the fluctuations in predictive performance over time, a fading factor is adopted as recommended for online learning (Gama et al. 2013).

9 Conclusion

We proposed novel online project-level CP approaches with the aim of pursuing a balance between diversity and validity of the selected OP data. These approaches aim to address the limitations of existing online data-level CP methods (i.e., AIO and Filtering). We conducted a thorough literature review for the collection of project-level similarity metrics. Based on the selection of these metrics, we formulated our project-level similarity measure, enabling us to quantify the instructiveness of each individual other project for the TP. Subsequently, we proposed CroPS and its enhanced version named Multi-CroPS, utilizing the CP similarity measure we formulated. These proposed online project-level CP approaches considered the balance between diversity and validity of OP data by incorporating project-level data selection into the learning procedures.

Our experimental results demonstrated the effectiveness of the designed project-level similarity in well capturing the instructiveness of each individual cross project. This indicates that the adoption of the proposed project-level similarity has the potential in providing great promise to the proposal of online CP methods. Our experimental results based on 23 open-source projects demonstrated the significant effectiveness of the proposed CroPS compared to WP and other online CP method, which could perform either significantly better than or on par with state-of-the-art online CP methods, indicating the ability to balance the diversity and validity of the selected OP data. Experimental results also demonstrated the significant superiority of the proposed Multi-CroPS over other online CP methods, including the proposed CroPS.

In addition, our experimental results in checking the average running time costs showed that the performance superiority of Multi-CroPS over CroPS and other online CP methods comes at the expense of higher computational costs, primarily due to the maintenance of multiple CroPS models simultaneously. Therefore, we would recommend adopting Multi-CroPS when sufficient computing resources are available and the timely response is not of an extreme demand. Conversely, in situations where computing resources are limited or an extremely prompt response is required, we would recommend using CroPS.

As for future work, it may be worth considering the proposal of an adaptive threshold for CroPS. By incorporating an adaptive threshold, the challenge of determining an optimal threshold for CroPS over time can be tackled, potentially leading to improved predictive performance. Moreover, the adoption of an adaptive threshold would streamline the approach by maintaining a single CroPS model, reducing computational costs compared to Multi-CroPS while still retaining the hope in achieving good predictive performance.

Funding This work was supported by State Key Laboratory of Robotics (Grant No. 2023-O11), National Natural Science Foundation of China (Grant Nos. 62002148 and 62250710682), Harbin Institute of Technology Talent Start-up Project (Grant No. AUGA5710010924), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), and Research Institute of Trustworthy Autonomous Systems (RITAS).

Data Availability A replication package can be found in the CroPS repository, https://github.com/TRebirthC/CroPS/tree/submit-v1. The datasets generated during and/or analyzed during the current study are available in the same repository.

Declarations

Conflicts of Interest The authors have no relevant financial or non-financial interests to disclose. The authors have no competing interests to declare that are relevant to the content of this article. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript. The authors have no financial or proprietary interests in any material discussed in this article.

References

Bludau P, Pretschner A (2022) PR-SZZ: How pull requests can support the tracing of defects in software repositories. IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, IEEE, Munich, Germany, pp 1–12

Cabral GG, Minku LL (2022) Towards reliable online just-in-time software defect prediction. IEEE Trans Softw Eng 49(3):1342–1358

Cabral GG, Minku LL, Shihab E, Mujahid S (2019) Class imbalance evolution and verification latency in just-in-time software defect prediction. In: 2019 IEEE/ACM 41st international conference on software engineering (ICSE), IEEE, pp 666–676

Cabral GG, Minku LL, Oliveira AL, Pessoa DA, Tabassum S (2023) An investigation of online and offline learning models for online just-in-time software defect prediction. Empir Softw Eng 28(5):121

Chicco D, Jurman G (2020) The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. BMC Genomics 21

Chicco D, Warrens M, Jurman G (2021) The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment. IEEE Access 9:78368–78381

Cho Y, Kwon JH, Ko IY (2018) Cross-sub-project just-in-time defect prediction on multi-repo projects. In: 6th International workshop on quantitative approaches to software quality, pp 2–9

Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20(1):37–46

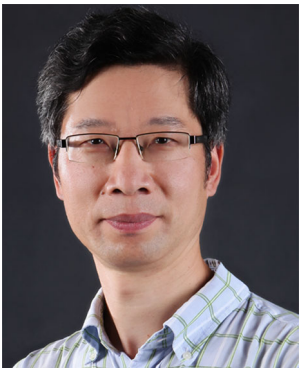
Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Favarò FM, Jackson DW, Saleh JH, Mavris DN (2013) Software contributions to aircraft adverse events: case studies and analyses of recurrent accident patterns and failure mechanisms. Reliab Eng Syst Saf 113:131–142
Fawcett T (2006) An introduction to roc analysis. Pattern Recognit Lett 27(8):861–874
Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi N (2014) An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the 11th working conference on mining software repositories, pp 172–181
Gama J, Sebastiao R, Rodrigues PP (2013) On evaluating stream learning algorithms. J Mach Learn 90(3):317–346
Hall T, Zhang M, Bowes D, Sun Y (2014) Some code smells have a significant but small effect on faults. ACM Trans Softw Eng Methodol 23(4)
Herbold S, Trautsch A, Trautsch F, Ledel B (2022) Problems with szz and features: an empirical study of the state of practice of defect prediction data collection. Empir Softw Eng 27(2)
Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N (2013) A large-scale empirical study of just-in-time quality assurance. IEEE Trans Softw Eng 39(6):757–773
Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE (2016) Studying just-in-time defect prediction using cross-project models. Empir Softw Eng 21:2072–2106
Kim S, Whitehead Jr EJ (2006) How long did it take to fix bugs? In: Proceedings of the 2006 international workshop on mining software repositories, pp 173–174
Kim S, Whitehead EJ, Zhang Y (2008) Classifying software changes: clean or buggy? IEEE Trans Softw Eng 34(2):181–196
Kubat M, Holte RC, Matwin S (1998) Machine learning for the detection of oil spills in satellite radar images. Mach Learn 30:195–215
Lin D, Tantithamthavorn C, Hassan AE (2021) The impact of data merging on the interpretation of cross-project just-in-time defect models. IEEE Transactions on Software Engineering 48(8):2969–2986
Matthews BW (1975) Comparison of the predicted and observed secondary structure of t4 phage lysozyme. Biochimica et Biophysica Acta (BBA)-Protein Structure 405(2):442–451
McIntosh S, Kamei Y (2018) Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. IEEE Trans Softw Eng 44(5):412–428
Mirjalili S, Mirjalili S (2019) Genetic algorithm. Theory and Applications, Evolutionary Algorithms and Neural Networks, pp 43–55
Newman M (2002) Software errors cost us economy $59.5 billion annually. NIST Assesses Technical Needs of Industry to Improve Software-Testing
Nugroho YS, Hata H, Matsumoto K (2020) How different are different diff algorithms in git? use-histogram for code changes. Empir Softw Eng 25:790–823
Obuchowski NA (2005) Roc analysis. Am J Roentgenol 184(2):364–372
Rezk C, Kamei Y, McIntosh S (2022) The ghost commit problem when identifying fix-inducing changes: an empirical study of apache projects. IEEE Trans Softw Eng 48(9):3297–3309
Rodríguez-Pérez G, Nagappan M, Robles G (2022) Watch out for extrinsic bugs! a case study of their impact in just-in-time bug prediction models on the openstack project. IEEE Trans Softw Eng 48(4):1400–1416
Rosen C, Grawi B, Shihab E (2015) Commit guru: analytics and risk prediction of software commits. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering, pp 966–969
Shehab MA, Hamou-Lhadj A, Alawneh L (2022) Clustercommit: a just-in-time defect prediction approach using clusters of projects. 2022 IEEE International Conference on Software Analysis. Evolution and Reengineering (SANER), IEEE, pp 333–337
Śliwerski J, Zimmermann T, Zeller A (2005) When do changes induce fixes? ACM SIGSOFT Softw Eng Notes 30(4):1–5
Song L, Minku LL (2023) A procedure to continuously evaluate predictive performance of just-in-time software defect prediction models during software development. IEEE Trans Softw Eng 49(2):646–666
Song L, Li S, Minku LL, Yao X (2022) A novel data stream learning approach to tackle one-sided label noise from verification latency. In: 2022 International joint conference on neural networks (IJCNN), IEEE, pp 1–8
Song L, Minku LL, Teng C, Yao X (2023a) A practical human labeling method for online just-in-time software defect prediction. In: Proceedings of the ACM joint european software engineering conference and symposium on the foundations of software engineering (ESEC/FSE), pp 605–617
Song L, Minku LL, Yao X (2023) On the validity of retrospective predictive performance evaluation procedures in just-in-time software defect prediction. Empir Softw Eng 28(5):1–33. https://doi.org/10.1007/s10664-023-10341-8

Tabassum S, Minku LL, Feng D, Cabral GG, Song L (2020) 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, pp 554–565
Tabassum S, Minku LL, Feng D (2022) Cross-project online just-in-time software defect prediction. IEEE Trans Softw Eng 49(1):268–287
Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalance data. In: 2015 IEEE/ACM 37th IEEE International conference on software engineering, IEEE, vol 2, pp 99–108
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2018) The impact of automated parameter optimization on defect prediction models. IEEE Trans Softw Eng 45(7):683–711
Wang S, Minku LL, Yao X (2015) Resampling-based ensemble methods for online class imbalance learning. IEEE Trans Knowl Data Eng 27(5):1356–1368
Wang S, Minku LL, Yao X (2018) A systematic study of online class imbalance learning with concept drift. IEEE Trans Neural Netw Learn Syst 29(10):4802–4821
Woolson RF (2007) Wilcoxon signed-rank test. Wiley encyclopedia of clinical trials pp 1–3
Yang X, Yu H, Fan G, Shi K, Chen L (2019) Local versus global models for just-in-time software defect prediction. Sci Program
Zhang T, Yu Y, Mao X, Lu Y, Li Z, Wang H (2022) Fense: a feature-based ensemble modeling approach to cross-project just-in-time defect prediction. Empir Softw Eng 27(7):162
Zheng S, Gai J, Yu H, Zou H, Gao S (2021) Training data selection for imbalanced cross-project defect prediction. Comput Electr Eng 94:107370
Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp 91–100
Zubrow D (2009) IEEE standard classification for software anomalies. IEEE Comput Soc

Cong Teng is a postgraduate student with the Southern University of Science and Technology, Shenzhen, China. He received his B.Sc degree in Computer Science from the same university in 2021. His main research interest includes classical machine learning techniques and their applications to Just-In-Time Software Defect Prediction, including online Cross-Project JIT-SDP and interpretable JIT-SDP.

Liyan Song received the bachelor's and master's degrees in mathematics from Harbin Institute of Technology (HIT), Harbin, China and the PhD degree in computer science from the University of Birmingham, Birmingham, U.K. He is an Associate Professor with the Faculty of Computing, HIT, Harbin, China. Prior to that, she was research assistant professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China. She was a Research Fellow with the School of Computer Science, the University of Birmingham, Birmingham, U.K. Her main research interests are machine learning for predictive modeling in software engineering. She also has particular research interests in machine learning areas such as online learning, class imbalance learning, ensemble learning, and deep learning.



Xin Yao (Fellow, IEEE) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technologies, Beijing, China, in 1985, and the Ph.D. degree from USTC, in 1990. He is the Tong Ting Sun chair professor of Machine Learning at Lingnan University, Hong Kong, and a part-time professor of Computer Science with the University of Birmingham, Birmingham, U.K. He was a Distinguished Lecturer of the IEEE Computational Intelligence Society (CIS). He served as the president (2014-15) of IEEE CIS and the editor-in-chief (2003-08) of IEEE Transactions on Evolutionary Computation. His major research interests include evolutionary computation, class imbalance learning, online learning, and their practical applications, particularly in the fields such as software engineering. His work won the 2001 IEEE Donald G. Fink Prize Paper Award; 2010, 2016, and 2017 IEEE Transactions on Evolutionary Computation Outstanding Paper Awards; 2011 IEEE Transactions on Neural Networks Outstanding Paper Award; 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist); and other best paper awards at conferences. He received a 2012 Royal Society Wolfson Research Merit Award, the 2013 IEEE CIS Evolutionary Computation Pioneer Award, and the 2020 IEEE Frank Rosenblatt Award.

Authors and Affiliations

Cong Teng[1,2] · Liyan Song[3] · Xin Yao[4]

✉ Liyan Song
songly@hit.edu.cn

✉ Xin Yao
xinyao@ln.edu.hk

Cong Teng
12132358@mail.sustech.edu.cn

1 Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China

2 Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China

3 Faculty of Computing, Harbin Institute of Technology, Harbin, China

4 School of Data Science, Lingnan University, Hong Kong SAR, China