Received: 6 December 2022 | Revised: 22 February 2023 | Accepted: 7 March 2023

DOI: 10.1002/smr.2563

RESEARCH ARTICLE - TECHNOLOGY

Journal of Software: Evolution and Process WILEY

Cross-version defect prediction using threshold-based active learning

Yuanqing Mei | Xutong Liu | Zeyu Lu | Yibiao Yang | Huihui Liu | Yuming Zhou

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Correspondence
Yibiao Yang and Yuming Zhou, State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China.
Email: yangyibiao@nju.edu.cn and zhouyuming@nju.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Numbers: 62172205, 62072194

Abstract

Because defects in software modules (e.g., classes) might lead to product failure and financial loss, software defect prediction enables us to better understand and control software quality. Software development is a dynamic evolutionary process that may result in data distributions (e.g., defect characteristics) varying from version to version. In this case, effective cross-version defect prediction (CVDP) is not easy to achieve. In this paper, we aim to investigate whether the defect prediction method of the threshold-based active learning (TAL) can tackle the problem of the different data distribution between successive versions. Our TAL method includes two stages. At the active learning stage, a committee of investigated metrics is constructed to vote on the unlabeled modules of the current version. We pick up the unlabeled module with the median of voting scores to domain experts. The domain experts test and label the selected unlabeled module. Then, we merge the selected labeled module and the remaining modules with pseudo-labels from the current version into the labeled modules of the prior version to form enhanced training data. Based on the training data, we derive the metric thresholds used for the next iteration. At the defect prediction stage, the iterations stop when a predefined threshold is reached. Finally, we use the cutoff threshold of voting scores, that is, 50%, to predict the defect-prone of the remaining unlabeled modules. We evaluate the TAL method on 31 versions of 10 projects with three prevalent performance indicators. The results show that TAL outperforms the baseline methods, including three variations methods, two common supervised methods, and the state-of-the-art method Hybrid Active Learning and Kernel PCA (HALKP). The results indicate that TAL can effectively address the different data distribution between successive versions. Furthermore, to keep the cost of extensive testing low in practice, selecting 5% of candidate modules from the current version is sufficient for TAL to achieve a good performance of defect prediction.

KEYWORDS
cross-version defect prediction (CVDP), median, threshold-based active learning (TAL)

1 | INTRODUCTION

1.1 | Motivation

Software Quality Assurance (SQA) is essential to the success of a software project. Timely detecting and fixing the defects before releasing the products is vital to SQA. Recently, software defect prediction approaches have been advanced by active learning techniques to help allocate limited SQA resources by predicting the defect-proneness of software modules. As a result, more resources and effort can be spent on modules that are likely to have defects.

Traditional cross-version defect prediction (CVDP) models are mainly built using historical data from past versions, then predict the defective modules on new versions. It is assumed that new versions share the same product/process characteristics with the same distribution and their defects can be predicted based on previous versions. However, software development is a dynamic evolution process.[1] Developers would modify or refactor existing modules, add new modules, and delete old modules. These operations could get rid of existing defects and introduce new defects, causing the distribution of defects to change between successive versions. In this case, a defect prediction model built on the labeled modules in a previous version may not be very effective for the unlabeled modules in the current version because the model does not consider the change of distribution.

To address this problem, researchers introduce active learning to CVDP. First, active learning is used to acquire a small number of unlabeled modules from the current version for labeling. Then, these labeled modules are merged into the previous version to obtain a labeled training set. This enhanced training set is expected to mitigate the difference of data distribution from the current version and hence can produce a better prediction model. In our study, the reasons for introducing active learning to the process of CVDP are summarized as follows. On the one hand, a regular model of defect prediction is usually built on the assumption that the training and test data are collected from the same distribution. In other words, it is difficult to achieve a good prediction performance by directly applying regular modeling techniques to build the prediction models based on data with different distributions. On the other hand, active learning can actively select a small of unlabeled modules from the current version and acquire the label information from domain experts. These selected labeled modules of the current version are merged into the prior version to form a mixed labeled training set. Hence, with the help of active learning, it is expected to mitigate the difference of data distribution between the mixed labeled training data and the remaining unlabeled test data.

Lu et al.[2] were the first to introduce active learning into CVDP by querying domain experts to identify the most valuable modules from the current version for labeling. However, the limitation is that "the selected candidate modules only rely on the classification model built on the labeled modules from the prior version," which "ignores the distribution information contained in the unlabeled modules of the current version."[1] To tackle this problem, Xu et al.[1] proposed Hybrid Active Learning and Kernel PCA (HALKP) to select the representative module for labeling. The hybrid measure $h(x_i)$ of module x_i is defined as $h(x_i) = f(x_i)^{\beta} d(x_i)^{1-\beta}$, where the β ($0 \le \beta \le 1$) is a controlling parameter toward the two measures: the uncertainty measure $f(x_i)$ and the information density measure $d(x_i)$. Xu et al.[1] selected the module $\widehat{x_i}$ with the maximum hybrid measure value as the candidate module. In their experiment, the optimal β value was determined by the F1 measure based on the label information of the modules in the current version. However, such label information is unavailable in advance practice. As a result, the promising performance of HALKP reported in Xu et al.[1] is indeed a theoretical optimal performance.

In this paper, we propose a threshold-based active learning (TAL) method, which can effectively exploit the selected modules and the remaining unlabeled modules in the current version for CVDP. Compared with HALKP, one prominent advantage of TAL is that it does not depend on test-set-label-dependent parameters. In TAL, each investigated metric is a learner, and a committee is designed by the combination of all metrics. Each member of the committee, that is, each metric, votes in turn on each module of the current version. For each metric, voting is conducted in the most straightforward way: An unlabeled module gains one point if it is greater than or equal to its threshold; otherwise, it gets zero point. After aggregating the scores of each metric with the reciprocal of *SLOC* value, we obtain the total voting score of each unlabeled module in the current version. The higher the voting score of committee members, the greater the confidence of all members that the module is defective; in contrast, the lower the voting score, the greater the confidence of all members that the module is not defective. The most representative module is selected based on the median of voting scores, because the module has the greatest uncertainty of defect-proneness or non-defect-proneness. In other words, the module with the median of voting scores is difficult to determine whether it is defect-prone or not. The reason is that it has a 50% probability of being defectiveness or non-defectiveness based on the voting score. Specifically, our TAL approach consists of two stages:

- At the active learning stage, pick up the unlabeled module with the median of voting scores to domain experts. The domain experts test and label the selected unlabeled module. Then, merge the labeled module and the remaining modules with pseudo-labels from the current version into the labeled modules of the prior version to form an enhanced training data. After that, the threshold values of investigated metrics used for the next iteration are calculated.
- At the defect prediction stage, the iterations stop when a predefined threshold is reached, such as 5%, 10%, 15%, or 20% labeled modules in all modules of the current version. Finally, TAL uses the cutoff threshold of the voting score, that is, 50%, to predict whether the remaining unlabeled modules are defect-prone or not.

The key to TAL is to select the module with the most uncertainty (i.e., the median value of voting score) for labeling from the unlabeled modules of the current version at the active learning stage. More specifically, TAL consists of two major selection strategies: uncertainty sampling and committee-based sampling.[3] On the one hand, the module with the median of voting scores has the greatest uncertainty of being defect-proneness or non-defect-proneness. On the other hand, the voting score is computed by all investigated metrics of a committee based on the threshold values.

We evaluate the TAL on 31 versions of 10 projects from a public dataset. The results of the experiment show that

1. TAL outperforms the baseline methods, including logistic regression, random forest, and the state-of-the-art method HALKP.
2. At the active learning stage, the selection strategy of TAL is superior to the other two strategies: selecting modules with the maximum or minimum of voting scores as the candidate modules, respectively.
3. At the defect prediction stage, the cutoff threshold of 50% used in TAL has the overall best performance of defect prediction compared with the other seven cutoff thresholds.

In practice, selecting 5% of candidate modules from the current version is sufficient for TAL to achieve a good performance in defect prediction.

1.2 | Contribution

As described above, Lu et al.[2] were the first to introduce active learning into CVDP by querying domain experts to identify the most valuable modules from the current version for labeling. Xu et al.[1] proposed HALKP to select the representative module for labeling. Li et al.[3] proposed sample-based software defect prediction with an active semi-supervised learning method that is able to sample the modules that are most helpful for learning a good prediction model. The sample selection strategies of their studies for labeling are mostly focused on one sample selection strategy, for example, uncertainty sampling strategy. However, different from their selection strategies, the proposed TAL consists of two major selection strategies: uncertainty sampling and committee-based sampling. In summary, the main contributions of our study are as follows:

1. We propose TAL to mitigate the distribution difference between successive versions. Our TAL method creatively uses the metric thresholds of a committee to carefully choose the uncertainty unlabeled modules from the current version for labeling, which is the combination of two major selection strategies: uncertainty sampling and committee-based sampling.
2. TAL and its variants (E-pseudo and E-prior) have different application scenarios to support diverse requirements of software projects in practice. When the prior versions of a target project are not available, we suggest using E-pseudo; when the only prior version of a target project is available, we advise performing TAL as well as E-prior; and when multiple prior versions of a target project are available, we propose to apply the TAL method in practice.
3. We evaluate TAL and its variants on 31 versions of 10 projects with three performance indicators. To promote reproducible research, all datasets and scripts are available in our replication package (https://github.com/meiyuanqing/Threshold-based-Active-Learning).

1.3 | Organization

The rest of this paper is organized as follows. Section 2 introduces the relate work. Section 3 describes the proposed TAL method. Section 4 presents the implementation of TAL. Section 5 reports the experimental results. Section 6 discusses the experimental results. Section 7 concludes the paper and outlines the directions for future work.

2 | RELATED WORK

2.1 | Metric thresholds in defect prediction

A software metric[4] is the quantification of a software characteristic, which can be used to find potential problems with quality improvement and provide a quantitative evaluation of software quality. Because each metric evaluates the module of the source code from a different aspect, we can discover the relationship between the quality level and the risk level of the code with the help of a metric threshold. In other words, deriving metric thresholds is to divide the space of a metric value into regions. Depending on the regions a metric value is in, we can make an informed quality assessment of the measured entity.[5]

In the field of threshold derivation of software metrics, unsupervised learning methods are mainly based on the distributions of metrics values, whereas supervised methods principally rest on the relationships between metrics values and defect-proneness. Specifically, the unsupervised learning methods, which in general start from the minimum value of each metric and search for a candidate threshold toward the large value, assume that the larger the value of each metric is, the more likely the class is to be defective, that is, each metric is positively correlated with defect-proneness. Based on this assumption, various unsupervised methods have been proposed to determine the metric thresholds according to different threshold criteria, such as Alves et al.'s ranking method,[6] Ferreira et al.'s method,[7] Vale et al.'s method,[8] and Oliveira et al.'s method.[9] For example, in the study of Alves et al.,[6] the measurement data for different software systems are pooled and aggregated, and the threshold's derivation method investigates a reasonable percentage of the source code volume and the metric's variability between systems. The segmented thresholds are derived by choosing the percentage, such as 70%, 80%, and 90%, of the overall code. To some extent, these unsupervised methods can also discriminate between defective and defect-free classes in practice, but their prediction performance is overall lower than that of supervised methods. For supervised methods, the prevailing approach is that the best candidate threshold of a given metric is determined by the maximum value of the defect prediction performance expressed by different indicators on the training dataset. These indicators include AUC (area under the ROC curve),[10] F-measure (F1),[11] balanced FPR–TPR (BPP)[12,13], and geometric mean (GM).[14]

Among these indictors in supervised methods, GM is one of the typical indicators. As highlighted in prior studies,[14,15] the effect of the negative cases (the clean modules) prevails in an imbalanced dataset. However, GM can keep a balance between the two accuracies: the true positive rate (TPR) and the true negative rate (TNR). That is, $GM = \sqrt{TPR \times TNR}$. Here, the TPR value is the ratio of the number of modules correctly classified as defect-prone to the total number of defective modules. The TNR indicates the rate of non-defect-prone modules that are classified correctly as non-defect-prone to the total number of non-defect-prone modules. For maximum GM (MGM) method, the criterion of the best candidate threshold is chosen to maximize the GM value. Because the GM indicator can keep the balance between the two accuracies and has been extensively used in previous studies,[14,16–19] we applied the MGM method based on the GM indicator in this study to derive the thresholds for each metric as well.

2.2 | Active learning

Active learning works with methods that consider the learner has some control over the input space to mitigate the difference of data distribution between the labeled and unlabeled data.[3] In the active learning process, a minimum number of queries on unlabeled data are expected to improve the learning performance. To exploit unlabeled data, an active learning method can acquire the ground-truth labels of specific examples from an oracle, for example, a domain expert. Recently, active learning has been introduced into the field of defect prediction.

2.2.1 | Sample selection strategy

The key component of active learning is the selection strategies for the appropriate unlabeled samples. There are two major strategies: uncertainty sampling and committee-based sampling.[3] For uncertainty sampling, the learner queries the unlabeled examples on which it is the least confident (Tong and Koller[20] and Balcan et al.[21]), whereas for committee-based sampling, a committee of multiple learners is designed to select the unlabeled examples on which the committee members disagree the most (Freund et al.[22]). To select uncertain samples, expected error reduction based active learning,[23] clustering-based active learning,[24,25] and informative and representative sample-based active learning[1,26] are proposed. In addition, Qu et al.[27] proposed a hybrid active learning query strategy using uncertainty sampling and query-by-committee for software defect prediction.[28] Li et al.[29] found that a meta-active learning query strategy could perform better than the commonly used query strategy when a little data was labeled.

2.2.2 | Sampling-based active learning

Li et al.[3] proposed an active semi-supervised learning method ACoForest to select the most helpful modules, which leveraged the advantages from both disagreement-based active learning and semi-supervised learning. They first used 10 different sampling rates {5%, 10%, 15%, ..., 50%} to initiate the ensemble learning model. They repeated the sampling process 100 times, and the average performance of the compared method was reported. Lu et al.[30] proposed an adaptive approach integrating supervised learning and active learning, where the Naïve Bayes classifier was used as the base learner. They also needed a certain number of labeled modules (samples) to initiate the methods. Luo et al.[31] introduced a two-stage active learning method combining the clustering and support vector machine techniques, which improved the performance of the predictor with less labeling effort. Mi et al.[32] also used clustering and active learning algorithms to filter and label representative data from the target versions. They argued that it solved the class imbalance problem in cross-project data and improved the defect prediction performance.

2.2.3 | Active learning in CVDP scenario

Lu et al.[2] were the first to introduce active learning into CVDP by querying domain experts to identify the most valuable modules from the current version for labeling and then merging them into the previous version to construct a hybrid training set.[1] Lu et al.[2] argued that the integration of active learning with uncertainty sampling consistently outperformed the corresponding supervised learning approach. They further improved the prediction performance with feature compression techniques, where feature selection or dimensionality reduction was applied to defect data prior to active learning. Inspired by the study of Lu et al.,[2] Xu et al.[1] proposed a two-phase CVDP framework HALKP, a hybrid active learning strategy HAL with a nonlinear feature extraction method kernel PCA. They evaluated HALKP on 31 versions of 10 projects from a public dataset, and the results showed that HALKP outperformed the state-of-the-art method proposed by Lu et al.[2] Because of the limitation of their methods as mentioned in Section 1.1, in this paper, we employ the metric thresholds to predict the defect-proneness of modules and obtain the voting scores of the unlabeled samples. Then, a committee of multiple learners is designed to select the unlabeled examples based on the composite voting score of the committee members with the most uncertain opinions.

2.3 | Software defect prediction

According to the availability of sufficient historical data for a project, software defect prediction can be grouped into within-project and cross-project defect prediction (CPDP).[1] For within-project defect prediction (WPDP), a software project has sufficient historical labeled data, and supervised defect prediction models are built to predict the defect-prone of the upcoming modules within the same project. For CPDP, a software project does not have sufficient labeled data. Hence, researchers proposed to transfer the knowledge from other projects, that is, source projects, to promote the prediction of the defect-prone modules in the current project, that is, target projects.[33]

To predict defect-prone modules in the target version of the project, it is most often to use a two-phase process, including model building and application.[34] In the model building phase, based on the metric data and the defect data (a.k.a. labeled data) collected from the modules in historical versions, a specific model is trained to construct the relationships between the metrics and defect-proneness. In the model application phase, based on the same metrics collected from the modules in the target version, the predicted defect-prone modules in the target version are achieved by replacing corresponding metric data in the model. Then, by comparing the predicted defect-prone modules with the actual defect-prone ones, researchers evaluate the predicted performance in the target version.

In the community of software defect prediction, the most essential challenge is that the source and target project data frequently exhibit significantly different distributions.[34–36] Because the source and target projects might be from different companies with different development environments, the same metric data and defect data respectively in the source and target projects might have an essential difference between the data distributions.[34] Yet, a specific model is built on the assumption that the training (source) and test (target) data are collected from the same distribution. More often than not, CPDP does not outperform WPDP in most cases, as the performance of the traditional CPDP approach is affected by the different data distributions between the source and target projects.[1,37] Hence, many studies exploited different techniques, such as ensemble techniques,[38] imbalance learning techniques,[39] and dynamic selection of learning techniques[40] to improve the performance of CPDP.

However, CVDP can be perceived as a special scenario between WPDP and CPDP.[1] For CVDP in a project with multiple versions, the labeled modules of the prior versions and the unlabeled modules of the current version can provide the source and target data, respectively. A better assistance for defect prediction is that the current version usually conserves much information from the prior versions. In other words, the distribution difference of the metric data and the defect data between the training data and test data would be smaller than that of across projects. Hence, CVDP will gain more practical benefits for defect prediction in the current version.

Recently, researchers introduced active learning to CVDP to address the problem of the distribution difference between source and target data. Xu et al.[1] proposed a state-of-the-art method, that is, a two-phase CVDP framework HALKP, and claimed that HALKP outperforms the method proposed by Lu et al.[2] Different from prior studies[41,42] that directly used the training data, that is, the metric data and defect data, of the prior version to predict the defect-proneness of modules in the current version, the approach of this kind of defect prediction framework is generally as follows.

First, based on the sample selection strategies, active learning is used to select the appropriate unlabeled samples from the current version. Then, domain experts are consulted to inspect the selected unlabeled modules and label these modules. After that, these selected labeled modules, which are the representative ones from the current version, are merged into the previous version(s) to obtain a mixed labeled training set. Finally, the enhanced mixed training set is expected to mitigate the distribution difference between the prior version(s) and the current version and contributes to a better predictive performance on remaining unlabeled modules of the current version.

In summary, active learning is explored in CVDP to mitigate the distribution difference of the metric data and the defect data between the training data and test data. Hence, this kind of method could contribute to better predictive performance against those methods in which active learning is not adopted.

3 | OUR METHOD

3.1 | Overview

In the initialization of our TAL, a committee is designed by the combination of all investigated metrics. Based on the threshold, each metric votes in turn on each unlabeled module of the current version. Aggregating the scores of each metric with the reciprocal of *SLOC* value, we obtain the total voting score of each unlabeled module in the current version. The initial threshold of each metric is the median value on the unlabeled modules of the current version. From the second iteration, the metric threshold is obtained from the mixed labeled modules pool of last iteration.

Figure 1 depicts the overview of our TAL framework that consists of two stages with the following seven steps. In each iteration of the first active learning stage, there are five steps: (1) ∼ (5), and the defect prediction stage includes step (6) and step (7).

1. The TAL method is used to select one unlabeled module that has the most uncertainty (i.e., the median of voting scores) from the unlabeled modules of the current version.
2. The domain experts, that is, verification engineers and software testers, are consulted to thoroughly inspect the selected module.
3. The domain experts assign the label to the selected module.
4. Pseudo-labels are assigned to the remaining unlabeled modules of the current version. That is, a defect-prone label (i.e., 1) is assigned to the unlabeled module if the voting score is greater than or equal to its median; otherwise, a non-defect-prone label (i.e., 0) is assigned.
5. The selected module with an assigned (real) label and the remaining modules with pseudo-labels of the current version are merged into the prior version to form the mixed labeled modules pool. According to the metric thresholds derived from the mixed labeled modules pool by the MGM method, the TAL method proceeds to the next iteration in the active learning stage.
6. The iterations stop when a predefined threshold is reached, such as 5%, 10%, 15%, or 20% labeled modules in all modules of the current version.
7. After the iteration, we select a cutoff threshold on the voting scores, that is, 50%, to predict defect-proneness of the remaining unlabeled modules. That is, the remaining unlabeled module of the current version is predicted as defect-prone if the voting score is greater than or equal to its cutoff threshold; otherwise, it is predicted as non-defect-prone.

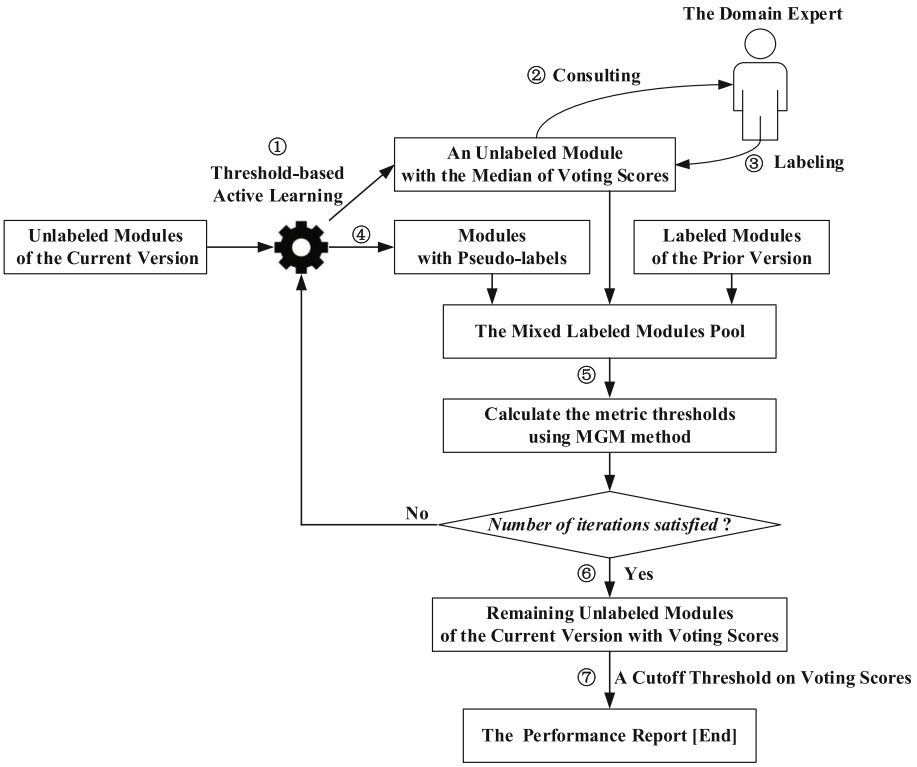

FIGURE 1 Overview of threshold-based active learning (TAL) framework.

To conclude, at the active learning stage, we pick up modules with real labels (labeled by experts) and the remaining modules with pseudo-labels from the current version. Then, we merge them into the labeled modules of the prior version to form an enhanced training data. After that, we calculate the threshold values of investigated metrics used for the next iteration. The key to the active learning stage is to select the module with the most uncertainty (the median of voting scores) for labeling from the unlabeled modules of the current version, which will be explained in detail in the next subsection. At the defect prediction stage, a cutoff threshold for the voting score is determined to predict the defect-prone of the remaining unlabeled modules of the current version.

3.2 | Voting score with uncertainty measure

To make better use of the labeled modules of the prior version and the distribution information of unlabeled modules (samples) of the current version, we select the most representative unlabeled module with the median of voting scores. In other words, we use the voting score measure to select the representative modules. The voting score measure $V(x_i)$ of module x_i is defined as in Equation (1),

$$V(xi) = \frac{1}{SLOC_{xi}} + \sum_{m \in M} S(m_{xi}, m_t) \tag{1}$$

where $S(m_{xi},m_t)$ equals to 1 in the case of $m_{xi} \geq m_t$, and $S(m_{xi}, m_t)$ equals to 0 in the case of $m_{xi} < m_t$, that is,

$$S(m_{xi}, m_t) = \begin{cases} 1 & \text{if } m_{xi} \geq m_t \\ 0 & \text{if } m_{xi} < m_t \end{cases} \tag{2}$$

In Equation (1), m_{xi} is the metric m value of an unlabeled module x_i in the current version, m_t is its threshold calculated on the mixed labeled module pool (the initial threshold is the median), $SLOC_{xi}$ is the *SLOC* value of an unlabeled module x_i, and m is in turn from all metrics in M, including *SLOC* metric. Here, each investigated metric m is a learner, and a committee is designed by the combination of all metrics in M. Each member of the committee, that is, each metric, votes in turn on each module of the current version. For each metric m, voting is conducted in the most straightforward way: An unlabeled module x_i gains one point if it is greater than or equal to its threshold m_t; otherwise, it gets zero point. After aggregating the scores of each metric m with the reciprocal of *SLOC* value, we obtain the total voting score $V(x_i)$ of each unlabeled module x_i in the current version. Based on the voting scores $V(x_i)$ ($i = 1, 2, \ldots, n$) on all modules, we select the most representative unlabeled module with the median of voting scores, which are given to domain experts for examination.

As described in Section 2.2, for uncertainty sampling, the learner queries the unlabeled examples on which it is the least confident (Tong and Koller[20] Balcan et al.[21]), whereas for committee-based sampling, a committee of multiple learners selects the unlabeled examples on which the committee members disagree the most (Freund et al.[22]). Intuitively, the higher the voting score of committee members, the greater the confidence of all members that the module is defective; otherwise, the lower the voting score, the greater the confidence of all members that the module is not defective. However, the module with the median voting score is difficult to determine whether it is defect-prone or not. The reason is that it has a 50% probability of defectiveness or non-defectiveness based on the voting score. Therefore, we select the module with the median of voting scores, which means that committee members have the least confidence in the defect-proneness or non-defect-proneness of the module. And the selected module with the remaining pseudo-label modules of the current version is merged into the labeled modules of the prior version for calculating the metric thresholds for the next iteration.

From Equation (1), it can be seen that the higher $V(x_i)$ of the module x_i, the more likely it is to be predicted as defect-prone; the lower $V(x_i)$ of the module x_i, the more likely it is to be predicted as non-defect-prone. According to the voting score with uncertainty measure, the module $\widehat{x_i}$ with the median of the voting measure $V(x_i)$ is selected as the candidate module, as shown in Equation (3).

$$\widehat{x_i} = \underset{x_i \in U}{argmedian} V(x_i) \tag{3}$$

where U is the unlabeled module set of the current version. A domain expert determines whether the selected module $\widehat{x_i}$ contains defects by thoroughly inspecting it and assigns it a label with their extensive experience. In this study, we simulate the process by assigning the ground-truth labels to the selected modules following prior work.[1–3]

Algorithm A Framework of threshold-based Active Learning.

Input: The labeled module set of the prior version, L;
The unlabeled module set of the current version, U;
Output: The mixed labeled module set of the prior and current version, L';
The rest of unlabeled module set of the current version, U'.

1: **repeat**
2: **for** $x_i \in U$ **do**
3: Calculate $V(x_i)$ using Eq. (1) based on the metric thresholds.
4: **end for**
5: Select a class x_i ($\widehat{x_i}$) that is the median value of voting measure using Eq. (3)
6: Remove $\widehat{x_i}$ from U ($U' = U - \{\widehat{x_i}\}$), $U = U'$.
7: Query the label $y(\widehat{x_i})$ of $\widehat{x_i}$.
8: Merge $(\widehat{x_i}, y(\widehat{x_i}))$ into L, $L' = L + (\widehat{x_i}, y(\widehat{x_i}))$.
9: Assign modules in U' with pseudo-labels, U'_{pseudo}.
10: Merge U'_{pseudo} into L' ($L'' = L' + U'_{\text{pseudo}}$), $L' = L''$.
11: Calculate the thresholds of all investigated metrics on the mixed module set L'.
12: **until** Meeting the stop criterion.
13: **return** L' and U'

Algorithm A presents the process of the TAL framework as depicted in Figure 1. After the active learning, we obtain two datasets, that is, the mixed labeled module set L' and the remaining unlabeled module set U'. The key to the voting score measure is to calculate the threshold values of the investigated metrics on the mixed labeled modules pool. In line 3 of Algorithm A, the initial threshold of each metric is the median value of all modules in U. From the second iteration, the metric threshold is derived from the mixed modules pool of last iteration using the MGM method (described in Section 2.1) as shown in line 11 of Algorithm A. In line 9 of Algorithm A, pseudo-labels are assigned to the remaining unlabeled modules in U' to form U'_{pseudo}. A defect-prone label (i.e., 1) is assigned to the unlabeled module in U' if the voting score is greater than or equal to its median; otherwise, a non-defect-prone label (i.e., 0) is assigned. In line 11 of Algorithm A, we calculate the thresholds of the investigated metrics, based on the mixed labeled module pool L'', including the labeled module set of the prior version (L), the labeled module $(\widehat{x_i}, y(\widehat{x_i}))$, and pseudo-label module set of U'_{pseudo}. When the selected modules reach a predefined threshold, the iterations stop. In our study, we use 5%, 10%, 15%, and 20% labeled modules in all modules of the current version as the predefined threshold. Note that the U'_{pseudo} will be identical to U' after the iterations stop. In other words, the pseudo-labels on the remaining unlabeled module are the final labels for defect prediction.

To better illustrate the framework of TAL, Figure 2 depicts one cycle of the TAL for selecting the module with the median of voting scores. In Figure 2, X_1 to X_7 represent metrics collected on the unlabeled modules *A* to *G* of the current version. Specifically, the following steps describe these in detail.

1. Determine the threshold of each metric.

In the initial cycle of active learning, we use the median value of each metric as the threshold. In the subsequent iteration, we use the MGM method to derive the threshold of each metric on the mixed labeled modules, including all labeled modules of the prior version, the selected modules with labels, and the remaining modules with pseudo-labels of the current version.

2. Voting on each module.

On each module, when a certain metric value is greater than or equal to its threshold value, a voting score of 1 point will be obtained; otherwise, no score will be obtained. In Figure 2, the metric values that are greater than the corresponding threshold are in bold font. To better distinguish each module, we add the reciprocal of *SLOC* metric value to the voting score of each module, because there may be modules with same voting score, such as module A and module E.

3. Select the module for active learning.

According to Equation (3), module D with the median of voting scores is selected for the next iteration, which is one part of the mixed labeled modules.

Unlabeled modules of current version

	X_1	X_2	X_3	X_4	X_5	X_6	X_7
Module A	$\mathbf{a_1}$	a_2	$\mathbf{a_3}$	a_4	a_5	a_6	$\mathbf{a_7}$
Module B	b_1	b_2	b_3	b_4	$\mathbf{b_5}$	b_6	b_7
Module C	$\mathbf{c_1}$	$\mathbf{c_2}$	c_3	$\mathbf{c_4}$	c_5	$\mathbf{c_6}$	c_7
Module D	d_1	d_2	$\mathbf{d_3}$	d_4	d_5	d_6	$\mathbf{d_7}$
Module E	e_1	e_2	e_3	$\mathbf{e_4}$	$\mathbf{e_5}$	$\mathbf{e_6}$	e_7
Module F	f_1	$\mathbf{f_2}$	f_3	f_4	f_5	f_6	f_7
Module G	g_1	g_2	g_3	g_4	g_5	g_6	g_7

Compare with the threshold of T_i

$X_i \geqslant T_i$	K	Voting score
$\mathbf{a_1}$, $\mathbf{a_3}$, $\mathbf{a_7}$	3	$3+(1/SLOC_A)$
$\mathbf{b_5}$	1	$1+(1/SLOC_B)$
$\mathbf{c_1}$, $\mathbf{c_2}$, $\mathbf{c_4}$, $\mathbf{c_6}$	4	$4+(1/SLOC_C)$
$\mathbf{d_3}$, $\mathbf{d_7}$	2	$2+(1/SLOC_D)$
$\mathbf{e_4}$, $\mathbf{e_5}$, $\mathbf{e_6}$	3	$3+(1/SLOC_E)$
$\mathbf{f_2}$	1	$1+(1/SLOC_F)$
/	0	$(1/SLOC_G)$

Prediction (Active learning)

Predictive labels (Real labels)
1
0
1
$\hat{x}_i$ (selected)
1
0
0

FIGURE 2 The unlabeled module selection strategy in threshold-based active learning (TAL).

4. Predict the defect-prone of each unlabeled module.

Except for the selected module for active learning, the remaining unlabeled modules will be predicted the defect-prone and assigned the 1 for pseudo-label when the voting score is greater than or equal to the median; otherwise, they will be predicted the non-defect-prone and assigned the 0 for pseudo-label. After the stage of active learning, these are the final defect predictions.

4 | IMPLEMENTATION

In this section, we tune the parameters in the TAL framework for CVDP.

1. The percent of modules in the current version selected for active learning.

In the phase of active learning, it costs to query the labels of the current version. However, if the procedure enables more effective defect prediction and improves the software quality, the cost is acceptable, as long as the number of queries is well limited to a small amount, normally within 20% of the total number of unlabeled modules.[1,2] In our experiment, we select four predefined thresholds, that is, 5%, 10%, 15%, and 20% of the total unlabeled modules, which are also used in Xu et al.[1] In practice, the labels of candidate unlabeled classes are assigned by the domain experts. Following previous work,[1–3] we simulate this process by assigning the ground-truth labels to candidate unlabeled modules.

2. The scenarios of CVDP.

Amasaki et al.[43] proposed three CVDP scenarios: SFV, SPV, and APV. In the scenario of SFV (Single Farthest Version), project data of the accessible farthest (oldest) prior version is used for building defect prediction. This setting can be suffered from data shift,[44] but it can save a maintenance cost of the prediction models. In the scenario of SPV (Single Prior Version), project data of the accessible latest prior version are used for that purpose. In the scenario of APV (All Prior Versions), project data of all accessible prior version are used. Following previous work,[1] we select SPV and APV as the CVDP scenarios.

3. The unlabeled modules sharing the same median value of voting scores.

As shown in Equation (3) and Figure 2, we add the reciprocal of *SLOC* metric value to the voting score to avoid the same voting score, such as modules A and E. In practice, it is also possible for modules A and E to have the same lines of code. In this case, we randomly select a module between A and E for the next iteration. The other module could be determined by the median voting score during the subsequent iterations. Indeed, this effect is marginal, because the number of modules selected for active learning remains the same in the end.

4. Filtering out learners with poor predictive performance in the committee.

We carefully examined the voting for each metric and found that the quality of voting for some metrics, especially the performance in predicting whether a module was defective, was quite poor. Therefore, to improve the quality of the voting scores, we filtered out one metric at a time in the committee starting from the second iteration. The filtering stops when the filtered metrics account for half of the number of members in the

committee. The criterion is that the metric which has the minimum value among the maximum values of GM in the process of calculating the threshold of this metric is filtered out. The reason is that this metric has the worst performance when its threshold alone is used to predict the module defect-prone. Although the maximum GM value of some metrics will get better in the next iterations, the difference of the maximum GM values between successive iterations is trivial to affect the voting score in subsequent iterations.

5 | EVALUATION

5.1 | Benchmark dataset

In our experiments, we used 31 versions of 10 software projects taken from the MORPH dataset. These datasets have been widely used in many previous studies.[1,45,46] Each module in the project contains 20 metrics and a dependent variable. These 20 metrics include weighted methods per class (*WMC*), Depth of Inheritance Tree (*DIT*), Number of Children (*NOC*), Coupling between object classes (*CBO*), Response for a Class (*RFC*), Lack

TABLE 1 Statistic of the benchmark dataset.

Project	Version	SLOC	Modules	Defective modules
Ant	1.3	38K	125	20 (16.00%)
	1.4	54K	178	40 (22.47%)
	1.5	87K	293	32 (10.92%)
	1.6	113K	351	92 (26.21%)
Camel	1.2	66K	608	216 (35.53%)
	1.4	98K	872	145 (16.63%)
	1.6	113K	965	188 (19.48%)
Ivy	1.1	27K	111	63 (56.76%)
	1.4	59K	241	16 (6.64%)
	2.0	88K	352	40 (11.36%)
Jedit	3.2	129K	272	90 (33.09%)
	4.0	145K	306	75 (24.51%)
	4.1	153K	312	79 (25.32%)
Log4j	1.0	22K	135	34 (25.19%)
	1.1	20K	109	37 (33.94%)
	1.2	38K	205	189 (92.20%)
Poi	1.5	55K	237	141 (59.49%)
	2.0	93K	314	37 (11.78%)
	2.5	120K	385	248 (64.42%)
Synapse	1.0	29K	157	16 (10.19%)
	1.1	42K	222	60 (27.03%)
	1.2	54K	256	86 (33.59%)
Velocity	1.4	52K	196	147 (75.00%)
	1.5	53K	214	142 (66.36%)
	1.6	57K	229	78 (34.06%)
Xalan	2.4	225K	723	110 (15.21%)
	2.5	305K	803	387 (48.19%)
	2.6	412K	885	411 (46.44%)
Xerces	1.2	159K	440	71 (16.14%)
	1.3	167K	453	69 (15.23%)
	1.4	141K	588	437 (74.32%)
Total	/	3214K	11537	3796 (32.9%)

of cohesion in methods (*LCOM* and *LCOM3*), Number of Public Methods (*NPM*), Data Access Metric (*DAM*), Measure of Aggregation (*MOA*), Measure of Functional Abstraction (*MFA*), Cohesion Among Methods of Class (*CAM*), Inheritance Coupling (*IC*), Coupling Between Methods (*CBM*), Average Method Complexity (*AMC*), Afferent couplings (*Ca*), Efferent couplings (*Ce*), McCabe's cyclomatic complexity (*Max_CC* and *Avg_CC*), and Lines of Code (*SLOC*). The dependent variable is the defect-proneness in each module, which is a binary variable, 0 for no-defect and 1 for one or more defects.

The datasets are summarized in Table 1. The statistics of the projects include the sum of non-commentary source lines of code in a module (*SLOC*), the number of modules, and the defective modules with the percentage in parentheses.

5.2 | Performance measures

5.2.1 | F1, GM, and BPP

We evaluate the performance of all the methods in terms of F1, GM, and BPP, which are widely used in defect prediction.[1,3,34,47] The three indicators are defined as follows.

$$\mathrm{F1} = 2 \times \mathrm{recall} \times \mathrm{precision} / (\mathrm{recall} + \mathrm{precision}) \tag{4}$$

$$\mathrm{GM} = \sqrt{\mathrm{TPR} \times \mathrm{TNR}} \tag{5}$$

$$\mathrm{BPP} = 1 - \sqrt{(0 - \mathrm{FPR})^2 + (1 - \mathrm{TPR})^2} / \sqrt{2} \tag{6}$$

In the equations of three indicators, TP, TN, FP, and FN are the modules actually defective and predicted to be defective, the modules actually non-defective and predicted to be non-defective, the modules actually non-defective and predicted to be defective, and the modules actually defective and predicted to be non-defective, respectively. The TPR, TNR, and FPR are TP/(TP + FN), TN/(TN + FP), and FP/(FP + TN), respectively. The recall and precision are TP/(TP + FN) and TP/(TP + FP), respectively.

5.2.2 | The Wilcoxon signed-rank test and Cliff's δ value

In our study, we use the Wilcoxon signed-rank test,[48,49] a nonparametric test, to analyze whether the differences between TAL and the baseline methods are statistically significant at the 95% confidence level. As it is a nonparametric test, this test is applicable for an unknown distribution. If the *p*-value of the Wilcoxon test is less than 0.05, the difference is significant; otherwise, not significant.

In addition, we use Cliff's δ value,[50,51] a nonparametric effect size measure, to quantify the magnitude of the difference. By convention,[51] the magnitude of the difference is considered trivial for $|\delta| < 0.147$, small for $0.147 \le |\delta| < 0.33$, moderate for $0.33 \le |\delta| < 0.474$, and large for $|\delta| \ge 0.474$. The Wilcoxon signed-rank Test is implemented by the wilcox.test function of the *rcompanion* R package. And Cliff's δ value is computed by the cliff.delta function of the *effsize* R package. (https://rcompanion.org/handbook/F_04.html)

5.2.3 | The Scott–Knott ESD test

We applied the Scott–Knott effect size difference test (v2.0) to compare the effect of different percentages of the cutoff threshold on prediction performance. The Scott–Knott ESD test (v2.0)[52] is a mean comparison approach that performs a hierarchical clustering to partition a set of treatment means, for example, means of predictive performance, into statistically distinct groups with non-negligible difference. According to the study of Tantithamthavorn et al.,[53] the mechanism of the Scott–Knott ESD test (v2.0) is composed of two steps:

a. Identify a partition that maximizes treatment means between groups by computing the sum of squares between groups;
b. Split into two groups or merge into one group by analyzing the magnitude of the difference for each pair for all of the treatment means of the two groups.

If there is one pair of two groups whose treatment means are non-negligible, we split into two groups. Otherwise, we merge into one group. Based on the study of Tantithamthavorn et al.,[53] the Scott–Knott ESD test (v2.0) is implemented by the sk_esd function of the ScottKnottESD R package.

5.3 | Experimental results

5.3.1 | RQ1: Is TAL more effective than other methods in CVDP scenarios?

Method

In the field of defect prediction, if multiple versions of a project are available, researchers proposed to conduct CVDP on the current version by utilizing the labeled modules of its prior versions.[2] Based on the features (input) and the labels (output) of the modules, supervised learning aims to learn a function that maps an input to an output.

In this question, we investigate whether our TAL method is more effective than the supervised methods and the state-of-the-art method HALKP in CVDP scenarios. The common supervised techniques include logistic regression (LR)[54–60] and random forest (RF),[61–68] which have been widely used in the field of defect prediction.

1. LR.

LR is a standard statistical modeling technique with a binary dependent variable. It is suitable for building defect prediction models as the functions under consideration are divided into two categories: defect-proneness and non-defect-proneness.[56] Let $Pr(Y = 1| X_1, X_2, ..., X_n)$ be the probability that the dependent variable $Y = 1$ given the independent variable X_1, X_2, ..., and X_n (i.e., the metrics in this study). Then, a multivariate LR model assumes that the $Pr(Y = 1|X_1, X_2, ..., X_n)$ is related to X_1, X_2, ..., and X_n according to the following equation:

$$Pr(Y=1|X1,X2,...,Xn)=\frac{e^{\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_nX_n}}{1+e^{\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_nX_n}} \tag{7}$$

As shown in Equation (7), $(n + 1)$ parameters of β_i in the model are the regression coefficients and can be estimated through the maximization of a log-likelihood based on the training data in the prior version(s). When the parameters of the model are estimated, we can use the model to predict the defect-proneness of the modules in the current version. We use the LogisticRegression function of the sklearn.linear_model package to obtain the defect prediction value of the current version. (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

2. RF.

An RF is a meta-estimator that fits a number of decision tree classifiers on various subsamples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. In other words, an RF is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.[68] Similar to the LR model, the RF model is built on the training data in the prior version(s) and the parameters of the model are estimated. Then, we can use the model to predict the defect-proneness of the modules in the current version.

If project data of the latest prior version are available, the RF model and the LR model are built in the scenario of SPV, whereas if project data of multiple prior versions are available, the RF model and the LR model are built in the scenario of APV. We use the RandomForestClassifier function of the sklearn.ensemble package to obtain the defect prediction value of the current version. (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

3. HALKP.

Recently, Xu et al.[1] designed a state-of-the-art method for active learning in the CVDP scenario and argued that it is superior to the state-of-the-art method proposed by Lu et al.[2] They proposed a two-phase CVDP framework that combines HALKP. Specifically, at the first stage, HALKP uses a hybrid active learning method to select some informative and representative unlabeled modules from the current version for querying their labels and then merges them into the labeled modules of the prior version to form an enhanced training set. At the second stage, HALKP employs a nonlinear mapping method, kernel PCA, to extract representative features by embedding the original data of two versions into a high-dimension space.

Because of the limitation of HALKP as mentioned in Section 1.1, our TAL does not depend on test-set-label-dependent parameters. In addition, compared with HALKP, TAL also consists of two stages, that is, the active learning stage and the defect prediction stage. However, TAL is simpler and more efficient than HALKP. On the one hand, TAL selects the module with the most uncertainty, that is, the median of voting scores, for labeling from the unlabeled modules of the current version. The voting score measure of unlabeled modules can be achieved by computation in Equation (1), which is simpler than the hybrid active learning method in HALKP. On the other hand, TAL applies a cutoff threshold for the

voting score to predict the defect-prone of the remaining unlabeled modules of the current version. The default value of the cutoff threshold is the median of the voting scores, which is more efficient than the nonlinear mapping method, kernel PCA, used in HALKP. Compared with LR and RF methods, TAL also utilizes the label data of prior version(s). Yet, TAL additionally selects a limited number of unlabeled modules from the current version and acquires label information through domain experts by active learning method. To this end, our TAL method can mitigate the distribution difference between the prior version(s) and the current version, thus potentially enhancing the defect prediction performance.

As Xu et al.[1] stated in their study, the experimental results of HALKP indicated that HALKP achieves encouraging results with average F1, GM, and BPP of 0.48, 0.592, and 0.58, respectively, and significantly outperforms nearly all their baseline methods. Hence, we evaluate our TAL framework on the same datasets of 31 versions from 10 projects to see if the three indicators are superior to those of the HALKP method.

Note that, because HALKP and TAL can actively select a number of informative unlabeled data for extensive software testing, it actually uses less the unlabeled data than the LR and RF method during the defect prediction stage. To achieve fair comparisons, for LR and RF methods, we conduct experiments with the same number of unlabeled modules used in TAL.

Results

Table 2 presents the results of three indicators under the predefined threshold of 5%. For each cross-version pair, the best values of the indicator are in bold. If two or more values are the same as the best one, they are all in bold. Note that the cutoff threshold of voting score in the defect prediction stage is 60% for the current versions of Ant-1.5, Ant-1.6, Ivy-2.0, and Jedit-4.1, whereas for others, the cutoff threshold is 50%. The reason for the cutoff threshold of 60% in these four versions is that we found that the prediction performance of the 60% threshold was better than that of the 50% threshold in the corresponding previous versions of these four versions. The details of this issue will be summarized in the results of RQ3. For the results of the four predefined thresholds, that is, 5%, 10%, 15%, and 20%, we present the corresponding comparison results as shown in Tables A.1, A.2, and A.3 of Appendix A.

From Table 2, we make the following observations.

1. In terms of F1, TAL achieves the best average performance over all three baseline methods. TAL outperforms the baseline methods on 17 out of 31 cross-version pairs. The average value of F1 by TAL is 0.525, which gains improvements of 50%, 37.8%, and 9.4% over the LR, RF, and HALKP methods, respectively. The *p*-values of the Wilcoxon test show that the difference between TAL and the three baseline methods is statistically significant ($p < 0.05$) except for the difference between TAL and HALKP ($p = 0.085$). But Cliff's δ value of comparison between TAL and HALKP in terms of F1 (0.161) is greater than 0.147, where the magnitude of the difference is not trivial (small). And Cliff's δ value between TAL and LR is 0.532, where the magnitude of the difference is large, and the magnitude of the difference between TAL and RF is moderate (0.382).
2. In terms of GM, the TAL method has a better performance of defect prediction compared with the three baseline methods, and the difference is all statistically significant. TAL is superior to the baseline methods on 20 out of 31 cross-version pairs. The average value of GM by TAL is 0.655, which gains improvements of 44.3%, 32.6%, and 10.6% over the LR, RF, and HALKP methods, respectively. The *p*-values of the Wilcoxon test show that the difference between TAL and the three baseline methods is statistically significant as well ($p < 0.05$). And Cliff's δ values show that the magnitude of the difference between TAL and LR (0.754) is large as well as the difference between TAL and RF (0.598), and the magnitude of the difference between TAL and HALKP is moderate (0.399).
3. In terms of BPP, the TAL method has a better performance of defect prediction compared with the three baseline methods, and the difference is all statistically significant with a large magnitude. TAL is better than the baseline methods on 24 out of 31 cross-version pairs. The average value of BPP by TAL is 0.65, which gains improvements of 38.9%, 30%, and 12.1% over the LR, RF, and HALKP methods, respectively. The *p*-values of the Wilcoxon test show that the difference between TAL and the three baseline methods is statistically significant as well ($p < 0.05$). And Cliff's δ values show that the magnitude of the difference between TAL and all three baseline methods are all large, where the δ values are 0.798, 0.609, and 0.48, respectively.

> **Finding 1**: The TAL method has better performance of defect prediction compared with the three baseline methods in terms of F1, GM, and BPP.

5.3.2 | RQ2: Why is the selection strategy of TAL better than the other two kinds of strategies?

Method

As mentioned in Section 3, TAL includes the active learning stage and the defect prediction stage. At the active learning stage, we select the module with the median of voting scores as the candidate module, as the module has the most uncertainty of defect-proneness or non-defect-proneness. The higher the voting score, the greater the confidence of all members of committee members that the module is defective; otherwise, the

TABLE 2 The results for threshold-based active learning (TAL) and three baseline methods under the predefined threshold of 5%.

Cross-version pair			F1				GM				BPP			
Project	Prior	Current	LR	RF	HALKP	TAL	LR	RF	HALKP	TAL	LR	RF	HALKP	TAL
Ant	1.3	1.4	0.207	0.197	0.359	**0.387**	0.369	0.364	0.538	**0.565**	0.395	0.393	0.527	**0.564**
	1.4	1.5	0.313	0.164	0.316	**0.338***	0.541	0.381	0.674	**0.707***	0.516	0.403	0.671	**0.703***
	1.5	1.6	0.353	0.248	0.383	**0.597***	0.478	0.390	0.53	**0.738***	0.461	0.405	0.51	**0.737***
	1.3 + 1.4 + 1.5	1.6	0.376	0.263	0.422	**0.640***	0.491	0.402	0.565	**0.771***	0.467	0.411	0.542	**0.770***
Camel	1.2	1.4	0.324	**0.501**	0.401	0.376	0.517	**0.735**	0.645	0.642	0.500	**0.732**	0.638	0.636
	1.4	1.6	0.176	0.340	**0.406**	0.361	0.325	0.493	**0.581**	0.575	0.370	0.476	0.557	**0.575**
	1.2 + 1.4	1.6	0.272	0.383	**0.429**	0.370	0.432	0.537	**0.583**	**0.583**	0.433	0.512	0.552	**0.582**
Ivy	1.1	1.4	0.170	0.155	0.165	**0.171**	0.596	0.480	0.583	**0.615**	0.594	0.456	0.583	**0.610**
	1.4	2.0	0.050	0.260	0.349	**0.384***	0.162	0.487	0.586	**0.756***	0.311	0.474	0.558	**0.741***
	1.1 + 1.4	2.0	0.338	0.294	0.3	**0.391***	0.530	0.484	0.583	**0.749***	0.503	0.468	0.564	**0.737***
Jedit	3.2	4.0	0.556	0.618	**0.634**	0.532	0.701	0.753	**0.772**	0.690	0.693	0.748	**0.769**	0.681
	4.0	4.1	0.583	0.586	**0.591**	0.574*	0.665	0.695	**0.732**	0.725*	0.617	0.666	0.724	**0.725***
	3.2 + 4.0	4.1	0.581	**0.614**	0.564	0.596*	0.692	0.722	0.708	**0.735***	0.667	0.700	0.699	**0.735***
Log4j	1.0	1.1	0.677	**0.719**	0.632	0.690	0.740	**0.774**	0.712	0.761	0.710	**0.750**	0.683	**0.750**
	1.1	1.2	0.444	0.443	**0.725**	0.671	0.498	0.476	0.401	**0.607**	0.487	0.475	0.41	**0.603**
	1.0 + 1.1	1.2	0.437	0.426	0.613	**0.669**	0.513	0.473	**0.622**	0.596	0.490	0.471	**0.598**	0.593
Poi	1.5	2.0	0.144	0.181	**0.259**	0.145	0.426	0.490	**0.585**	0.461	0.429	0.489	**0.584**	0.461
	2.0	2.5	0.065	0.178	0.604	**0.624**	0.183	0.307	**0.616**	0.572	0.317	0.363	**0.606**	0.572
	1.5 + 2.0	2.5	0.178	0.259	0.675	**0.713**	0.307	0.375	0.683	**0.685**	0.363	0.399	0.667	**0.682**
Synapse	1.0	1.1	0.286	0.152	0.426	**0.506**	0.426	0.292	0.574	**0.647**	0.428	0.355	0.559	**0.644**
	1.1	1.2	0.434	0.426	0.479	**0.562**	0.549	0.538	0.591	**0.646**	0.527	0.513	0.584	**0.644**
	1.0 + 1.1	1.2	0.421	0.379	0.458	**0.581**	0.527	0.498	0.568	**0.664**	0.499	0.480	0.544	**0.662**
Velocity	1.4	1.5	0.770	**0.790**	0.751	0.647	0.204	0.267	0.493	**0.596**	0.322	0.345	0.488	**0.596**
	1.5	1.6	0.593	**0.615**	0.596	0.571	0.600	0.641	**0.657**	0.651	0.569	0.608	0.647	**0.649**
	1.4 + 1.5	1.6	0.550	**0.593**	0.527	0.587	0.374	0.596	0.494	**0.659**	0.394	0.570	0.485	**0.657**

TABLE 2 (Continued)

Cross-version pair			F1				GM				BPP			
Project	**Prior**	**Current**	**LR**	**RF**	**HALKP**	**TAL**	**LR**	**RF**	**HALKP**	**TAL**	**LR**	**RF**	**HALKP**	**TAL**
Xalan	2.4	2.5	0.231	0.209	0.345	**0.581**	0.362	0.341	0.454	**0.585**	0.389	0.379	0.452	**0.585**
	2.5	2.6	0.517	**0.678**	0.639	0.545	0.568	**0.683**	0.671	0.556	0.564	**0.682**	0.668	0.556
	2.4 + 2.5	2.6	0.300	**0.656**	0.61	0.552	0.421	**0.693**	0.636	0.567	0.423	**0.684**	0.636	0.567
Xerces	1.2	1.3	0.184	0.190	0.294	**0.393**	0.324	0.385	0.469	**0.677**	0.368	0.406	0.457	**0.663**
	1.3	1.4	0.224	0.192	0.536	**0.777**	0.355	0.326	0.584	**0.779**	0.382	0.368	0.555	**0.748**
	1.2 + 1.3	1.4	0.086	0.103	0.402	**0.755**	0.212	0.232	0.468	**0.744**	0.325	0.332	0.466	**0.726**
Avg.			0.350	0.381	0.480	**0.525**	0.454	0.494	0.592	**0.655**	0.468	0.500	0.580	**0.650**
Wilcoxon test's *p*-value			<0.001	0.004	0.085		<0.001	<0.001	0.007		<0.001	<0.001	0.002	
Cliff's δ value			0.532	0.382	0.161		0.754	0.598	0.399		0.798	0.609	0.480	

*For Ant-1.5, Ant-1.6, Ivy-2.0, and Jedit-4.1, the cutoff threshold of prediction is 60%; for others, the cutoff threshold is 50%.

lower the voting score, the greater the confidence that the module is not defective. In this question, we investigate whether the other two kinds of strategies impact our experimental results in which we select the modules with the most certainty of defect-proneness and non-defect-proneness as the candidate modules for labeling, respectively. For this purpose, we design two baseline methods. On the one hand, we select the module with the maximum value of voting score as the candidate module, where the module has the most certainty of defect-proneness by members of the committee. We name this baseline method as TAL-max. On the other hand, we select the module with the minimum value of voting score as the candidate module, where the module has the most certainty of non-defect-proneness by members of the committee. We name this baseline method as TAL-min.

Results

Figure 3 depicts the box plots of TAL on three indicators under four predefined thresholds. As shown in Figure 3, we observe that the median of the indicators marked by the green line gradually increases as the predefined threshold increases. Similarly, the average value of the indicators marked by the green triangle moderately increases as the predefined threshold increases.

Furthermore, we observe that the average values of the three indicators under the predefined threshold of 5% have no big difference compared with the corresponding average values under the predefined threshold of 20%, as the average values of F1, GM, and BPP under the predefined threshold of 5% are 0.525, 0.655, and 0.650, respectively, whereas the average values of those under the predefined threshold of 20% are 0.543, 0.664, and 0.658, respectively.

Because a minimum number of queries is expected to improve the performance during the active learning process, selecting 5% of the candidate modules in a certain current version for labeling may be sufficient for TAL to achieve acceptable performance. In addition, it is usually difficult to conduct costly extensive testing for many modules of large projects in practice.

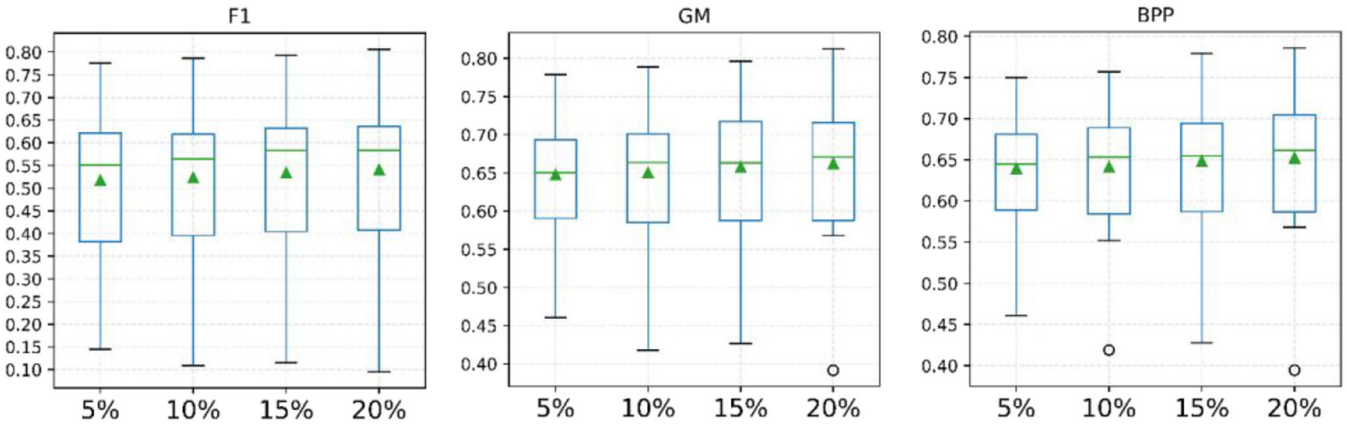

FIGURE 3 Box plots of threshold-based active learning (TAL) on three indicators under four predefined thresholds.

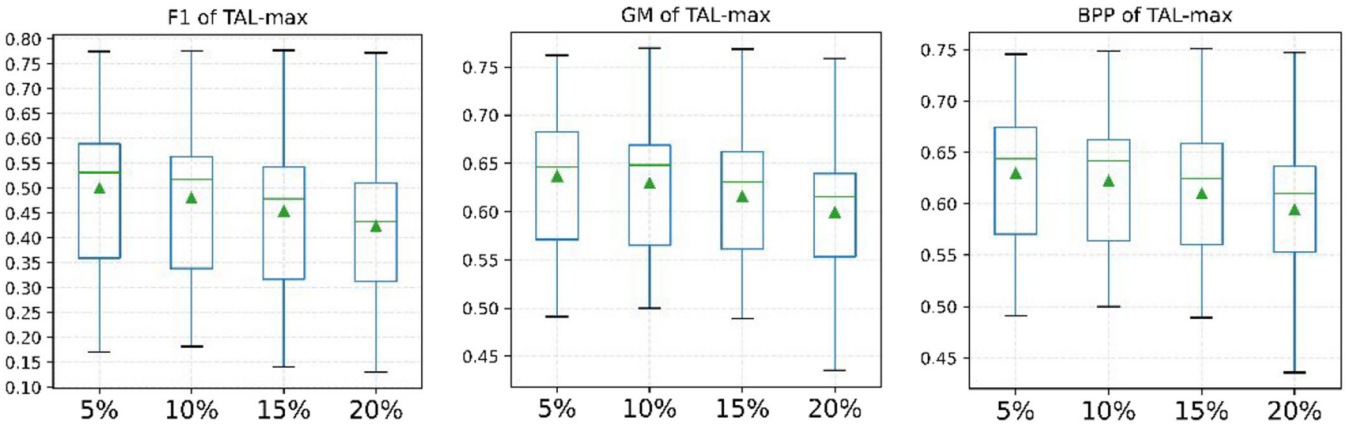

FIGURE 4 Box plots of TAL-max method on three indicators under four predefined thresholds.

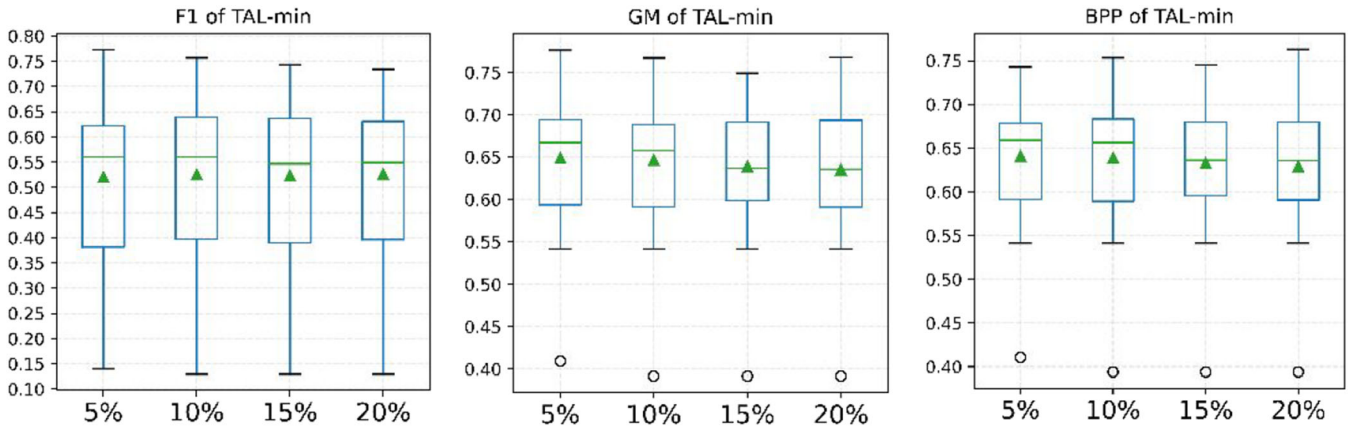

FIGURE 5 Box plots of TAL-min method on three indicators under four predefined thresholds.

Finding 2: Selecting five percent of candidate modules from the current version is sufficient for TAL to achieve good performance of defect prediction.

Figures 4 and 5 depict the box plots of three indicators for the two kinds of strategies under four predefined thresholds: TAL-max and TAL-min, respectively.

As shown in Figure 4, we observe that the median value of the indicators marked by the green line gradually decreases as the predefined threshold value increases except for GM at the threshold of 10%. The median value of GM under the four predefined thresholds (i.e., 5%, 10%, 15%, and 20%) are 0.647, 0.648, 0.631, and 0.616, respectively. Similarly, the average value of the indicators marked by the green triangle decreases as the predefined threshold value increases.

As shown in Figure 5, we observe that the median value of the indicators marked by the green line gradually decreases as the predefined threshold value increases except for F1 at the threshold of 10%. The median values of F1 under the four predefined thresholds are 0.560, 0.561, 0.547, and 0.549, respectively. The average value of the indicators marked by the green triangle decreases as the predefined threshold increases except for F1 at four thresholds. The average values of F1 under the four predefined thresholds are 0.521, 0.526, 0.524, and 0.527, respectively.

To conclude, the unlabeled module selection strategy of TAL is to select the median of voting scores for labeling. Indeed, the module with the median voting score is difficult to determine whether it is defect-prone or not, because there is a 50% probability of defectiveness or non-defectiveness. The sample (module) of this uncertainty is selected and used for active learning to improve the prediction performance of each committee member's threshold in subsequent iterations, and ultimately to enhance the prediction performance of the whole committee. Because the median or average value of almost all indicators gradually increases as the predefined threshold value increases as shown in Figure 3, the select strategy of TAL (the median of voting scores) is better than the other two select strategies, that is, TAL-max and TAL-min.

Finding 3: The select strategy of TAL is better than the other two strategies, that is, TAL-max and TAL-min.

5.3.3 | RQ3: How do the other eight kinds of *cutoff* thresholds impact the performance of TAL?

Method

At the defect prediction stage of TAL, a cutoff threshold for the voting score is determined to predict the defect-proneness of the remaining unlabeled modules of the current version. In our experiments, we use the median value of the voting score as the main cutoff threshold, where the 60% percentile of voting score is also used as the cutoff threshold in seven out of 31 cross-version pairs. In this question, we investigate whether the other kinds of cutoff thresholds impact our experimental results. For this purpose, we select the other eight percentiles of voting score as the cutoff thresholds, that is, 10%, 20%, 30%, 40%, 60%, 70%, 80%, and 90%, where we name these cutoff thresholds as P10, P20, P30, P40, P60, P70, P80, and P90, respectively.

For the convenience of comparison, the median cutoff threshold used in the TAL method is called P50. Then, we conduct the same experimental steps of TAL for the other eight cutoff thresholds to observe the difference of the defect prediction performance between TAL and the baseline methods.

Results

Figure 6 depicts the line plots of nine cutoff thresholds on three indicators under the predefined threshold of 5%, where the diamond point corresponding to each cutoff threshold is the average value of the indicator over 31 cross-version pairs.

As shown in Figure 6, we observe that the average values of both GM and BPP marked with red and green lines, respectively, increase as the cutoff threshold value increases from P10 to P60, whereas the average values of both GM and BPP decrease as the cutoff threshold value increases from P60 to P90. In terms of F1, we observe that the average values have no big difference as the cutoff threshold value increases from P10 to P60, whereas the average value decreases as the cutoff threshold value increases from P60 to P90. In summary, in terms of all three indicators, there is no big difference between the cutoff threshold of P50 and P60. The detailed results of Figure 6 inspire us that better prediction performance can be obtained by fine tuning the cutoff threshold on the voting score at the defect prediction stage of TAL. This is the reason why we fine tune the cutoff thresholds at 60% percentile on the current versions of Ant-1.5, Ant-1.6, Ivy-2.0, and Jedit-4.1 based on the prediction performance of corresponding previous versions.

We present the results of the Scott–Knott ESD test as shown in Figure 7 to visualize the difference in the average value of all three indicators between all cutoff thresholds under the predefined threshold of 5% modules from the current version. Circle dots in Figure 7 indicate the average values of the indicator for nine kinds of cutoff thresholds including P50 of TAL, respectively, and the lines show the 95% confidence interval of the indicator values.

As shown in Figure 7, each group in the Scott–Knott ESD test has a different ranking and is displayed in a different color. Specifically, those marked in black, red, green, blue, cyan, purple, and yellow are the first, second, third, fourth, fifth, sixth, and seventh ranking groups, respectively.

From Figure 7, we observe that the results of the Scott–Knott ESD test have seven groups of the means in terms of both GM and BPP, whereas the result has four groups of the means in terms of F1. For each indicator, the magnitude of the difference in each group is trivial, and the magnitude of the difference between groups is non-negligible. Specifically, in terms of F1, the cutoff thresholds of P1 to P60 belong to the first ranking group, and P70, P80, and P90 belong to the second, third, and fourth ranking groups that consist of only one element, respectively. In terms of GM and BPP, the cutoff thresholds of P50 and P60 belong to the first-ranking group; P40 and P70 belong to the second-ranking groups; and P30, P80, P20, P90, and P10 belong to the third, fourth, fifth, sixth, and seventh ranking groups that consist of only one element, respectively.

To conclude, the reliability of TAL at the stage of defect prediction depends on the selection of the cutoff threshold on the voting scores. As shown in Figure 7, there is no big difference between the cutoff threshold of P50 and P60 in terms of average values of all three indicators. In addition, it is due to the excellent performance of the median threshold that the TAL method selects the median threshold for pseudo-labeling at each iteration of the active learning stage, which is also the median value of the voting scores on the unlabeled modules of the current version.

> **Finding 4**: The cutoff threshold of 50% used in TAL has the best performance of defect prediction as well as the cutoff threshold of 60% compared with the other seven cutoff thresholds. This indicates that TAL has better prediction performance by fine tuning the cutoff threshold between 50% and 60% on the voting scores at the defect prediction stage.

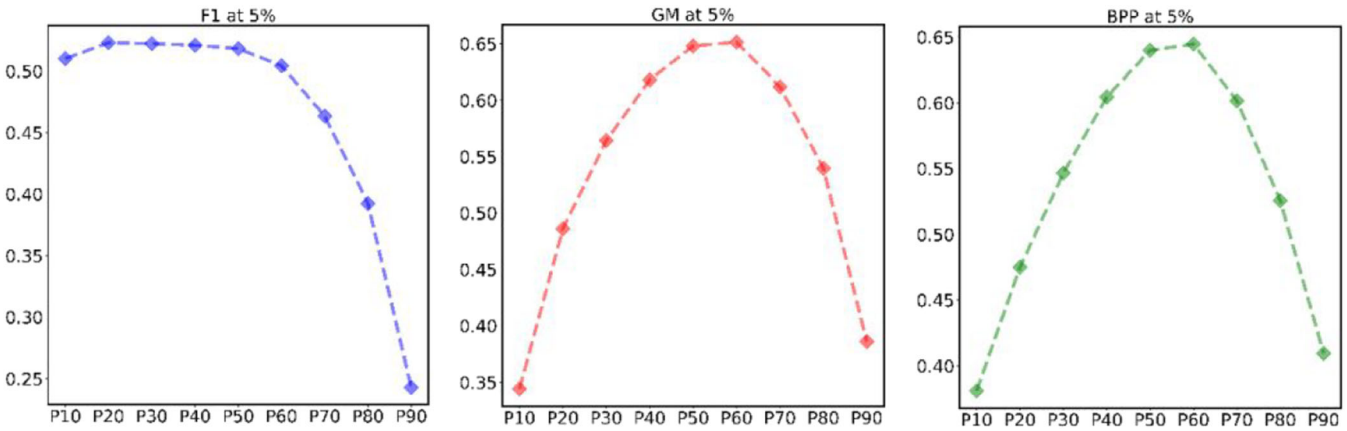

FIGURE 6 Line plots of nine cutoff thresholds on three indicators under the predefined threshold of 5%.

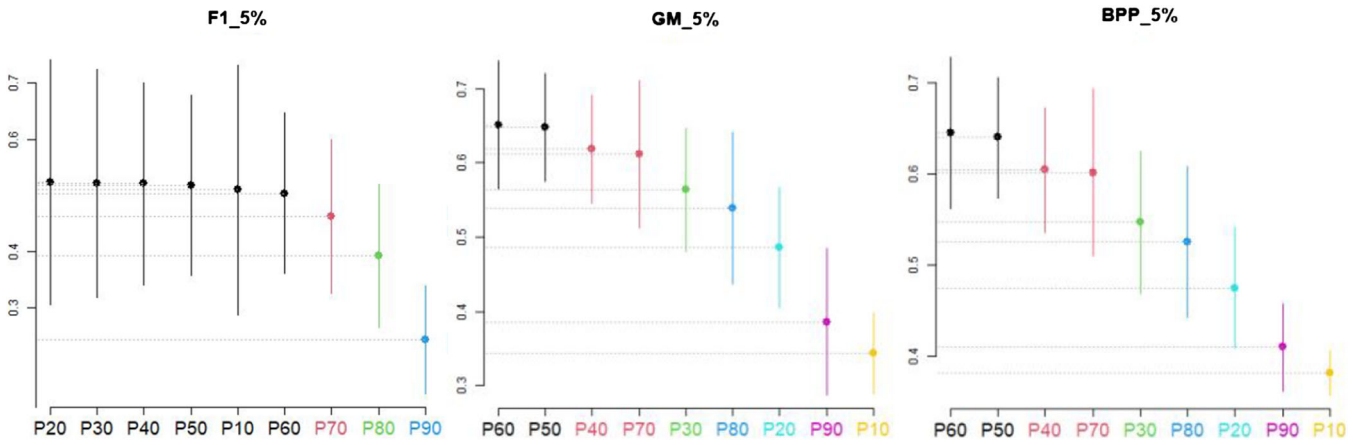

FIGURE 7 The Scott–Knott ESD ranking of the means on the three indicators under the predefined threshold of 5%.

6 | DISCUSSION

6.1 | TAL and its three variations with their possible application scenarios

As mentioned in Section 3, the voting score is calculated based on the thresholds of all metrics derived from the mixed labeled module pool, which includes three parts: the labeled modules of the prior version, the labeled modules, and pseudo-label modules of the current version.

In this subsection, we investigate whether the number of the mixed labeled modules impacts our experimental results at the active learning stage of TAL. For this purpose, we select three variants of TAL's downgraded methods based on three subsets of the mixed module pool. Then, we evaluate the performance of the three baseline methods of the downgraded variations. Finally, we summarize the possible application scenarios for TAL and its downgraded variation methods in practice.

These three baseline methods of the downgraded variations are described as follows.

1. *E-only*: the method applies the TAL method *only* by using the selected modules of the current version labeled by domain experts to derive the threshold value of each metric. The derived thresholds are used to calculate the voting scores for the next iteration.
2. *E-prior*: the method applies the TAL method by using the selected modules of the current version labeled by domain experts and the labeled modules of the *prior* version to derive the threshold value of each metric.
3. *E-pseudo*: the method applies the TAL method by using the selected modules labeled by domain experts and the *pseudo-labeled* modules of the current version to derive the threshold value of each metric.

According to Finding 2 above, we only report the detailed results of the three indicators for each cross-version pair under the predefined threshold of 5%. Tables 3 and 4 show the results for TAL and its three variations on SPV and APV scenarios, respectively. For each cross-version pair, the best values of the indicator are in bold as well. If two or more values are the same as the best one, they are all in bold.

From Tables 3 and 4, we make the following observations.

1. Compared with the E-only method in the SPV scenario.

As shown in Table 3, the average F1 of TAL gains improvements of 18.1% over that of the E-only method; the average GM of TAL is 15.6% greater than that of the E-only method; and the average BPP of TAL achieves improvements of 14.7% over that of the E-only method. In terms of all three indicators, TAL is superior to the E-only method on 18 out of 21 cross-version pairs except for two pairs in the Poi project and one pair in the Velocity project (Velocity-1.4 to Velocity-1.5). The *p*-values of the Wilcoxon test indicate that the performance differences between TAL and E-only are statistically significant. The magnitude of the difference measured by Cliff's δ value (0.256) is small in terms of F1, whereas the magnitude of the difference is large in terms of both GM and BPP (0.583 and 0.58).

2. Compared with the E-prior and E-pseudo methods in the SPV scenario.

As shown in Table 3, the average values of F1, GM, and BPP from the TAL method have no big difference compared to the E-prior and E-pseudo methods in the SPV scenario. Hence, the *p*-values of the Wilcoxon test show the difference between TAL and E-prior are not statistically

TABLE 3 The results for threshold-based active learning (TAL) and its three variations in the Single Prior Version (SPV) scenario under the predefined threshold of 5%.

Cross-version pair			F1				GM				BPP			
Project	Prior	Current	E-only	E-prior	E-pseudo	TAL	E-only	E-prior	E-pseudo	TAL	E-only	E-prior	E-pseudo	TAL
Ant	1.3	1.4	0.311	**0.435**	0.387	0.387	0.500	**0.612**	0.565	0.565	0.500	**0.610**	0.564	0.564
	1.4	1.5	0.212	0.304	0.304	**0.338***	0.543	0.662	0.662	**0.707***	0.543	0.648	0.648	**0.703***
	1.5	1.6	0.468	**0.609**	0.601	0.597*	0.634	**0.743**	**0.743**	0.738*	0.631	0.726	0.725	**0.737***
Camel	1.2	1.4	0.320	0.373	**0.383**	0.376	0.562	0.631	**0.648**	0.642	0.561	0.626	**0.641**	0.636
	1.4	1.6	0.282	**0.364**	0.351	0.361	0.499	**0.585**	0.570	0.575	0.499	**0.584**	0.569	0.575
Ivy	1.1	1.4	0.126	0.185	**0.200**	0.171	0.559	0.624	**0.652**	0.615	0.558	0.617	**0.638**	0.610
	1.4	2.0	0.254	0.340	0.330	**0.384***	0.599	0.704	0.692	**0.756***	0.596	0.675	0.669	**0.741***
Jedit	3.2	4.0	0.457	0.528	0.521	**0.532**	0.615	**0.693**	0.676	0.690	0.614	**0.683**	0.669	0.681
	4.0	4.1	0.372	0.518	0.566	**0.574***	0.526	0.672	0.710	**0.725***	0.526	0.666	0.699	**0.725***
Log4j	1.0	1.1	0.636	**0.698**	0.690	0.690	0.707	**0.775**	0.761	0.761	0.702	**0.760**	0.750	0.750
	1.1	1.2	0.633	0.669	0.667	**0.671**	0.428	0.596	0.585	**0.607**	0.429	0.593	0.583	**0.603**
Poi	1.5	2.0	**0.216**	0.205	0.177	0.145	0.537	**0.515**	0.499	0.461	**0.536**	0.515	0.499	0.461
	2.0	2.5	0.719	0.612	**0.745**	0.624	0.693	0.560	**0.724**	0.572	0.690	0.560	**0.720**	0.572
Synapse	1.0	1.1	0.405	**0.547**	0.472	0.506	0.547	**0.689**	0.619	0.647	0.547	**0.682**	0.618	0.644
	1.1	1.2	0.320	**0.562**	0.547	**0.562**	0.414	**0.646**	0.638	**0.646**	0.414	**0.644**	0.637	**0.644**
Velocity	1.4	1.5	**0.652**	0.525	0.632	0.647	**0.606**	0.433	0.579	0.596	**0.606**	0.434	0.579	0.596
	1.5	1.6	0.385	**0.602**	0.552	0.571	0.478	**0.667**	0.637	0.651	0.478	**0.666**	0.635	0.649
Xalan	2.4	2.5	0.566	**0.585**	0.576	0.581	0.573	**0.589**	0.583	0.585	0.573	**0.589**	0.583	0.585
	2.5	2.6	0.534	**0.644**	0.545	0.545	0.547	**0.655**	0.558	0.556	0.547	**0.655**	0.558	0.556
Xerces	1.2	1.3	0.252	**0.400**	0.339	0.393	0.507	**0.685**	0.634	0.677	0.507	**0.669**	0.628	0.663
	1.3	1.4	0.712	0.750	**0.790**	0.777	0.652	0.736	**0.795**	0.779	0.651	0.720	**0.760**	0.748
Avg.			0.421	0.498	0.494	0.497	0.558	0.642	0.644	0.645	0.558	0.634	0.637	0.640
Wilcoxon test's *p*-value			0.001	0.641	0.214		0.001	0.970	0.268		0.001	0.681	0.165	
Cliff's δ value			0.256	<0.001	0.036		0.583	-0.009	0.018		0.580	-0.009	0.050	

*For Ant-1.5, Ant-1.6, Ivy-2.0, and Jedit-4.1, the cutoff threshold of prediction is 60%; for others, the cutoff threshold is 50%.

TABLE 4 The results for threshold-based active learning (TAL) and its three variations in the All Prior Versions (APV) scenario under predefined threshold of 5%.

Cross-version pair			F1				GM				BPP			
Project	**Prior**	**Current**	**E-only**	**E-prior**	**E-pseudo**	**TAL**	**E-only**	**E-prior**	**E-pseudo**	**TAL**	**E-only**	**E-prior**	**E-pseudo**	**TAL**
Ant	1.3 + 1.4 + 1.5	1.6	0.468	0.627	0.601	**0.640***	0.634	0.762	0.743	**0.771***	0.631	0.740	0.725	**0.770***
Camel	1.2 + 1.4	1.6	0.282	**0.372**	0.351	0.370	0.499	**0.589**	0.570	0.583	0.499	**0.588**	0.569	0.582
Ivy	1.1 + 1.4	2.0	0.254	0.357	0.330	**0.391***	0.599	0.718	0.692	**0.749***	0.596	0.683	0.669	**0.737***
Jedit	3.2 + 4.0	4.1	0.372	0.522	0.566	**0.596***	0.526	0.669	0.710	**0.735***	0.526	0.664	0.699	**0.735***
Log4j	1.0 + 1.1	1.2	0.633	**0.674**	0.667	0.669	0.428	**0.617**	0.585	0.596	0.429	**0.611**	0.583	0.593
Poi	1.5 + 2.0	2.5	0.719	0.678	**0.745**	0.713	0.693	0.642	**0.724**	0.685	0.690	0.641	**0.720**	0.682
Synapse	1.0 + 1.1	1.2	0.320	0.547	0.547	**0.581**	0.414	0.638	0.638	**0.664**	0.414	0.637	0.637	**0.662**
Velocity	1.4 + 1.5	1.6	0.385	0.552	0.552	**0.587**	0.478	0.637	0.637	**0.659**	0.478	0.635	0.635	**0.657**
Xalan	2.4 + 2.5	2.6	0.534	0.549	0.545	**0.552**	0.547	0.561	0.558	**0.567**	0.547	0.561	0.558	**0.567**
Xerces	1.2 + 1.3	1.4	0.712	0.751	**0.790**	0.755	0.652	0.740	**0.795**	0.744	0.651	0.722	**0.760**	0.726
Avg.			0.468	0.563	0.569	0.585	0.547	0.657	0.665	0.675	0.546	0.648	0.656	0.671
Wilcoxon test's *p*-value			0.004	0.020	0.160		0.004	0.049	0.322		0.004	0.027	0.193	
Cliff's δ value			0.420	0.190	0.170		0.700	0.180	0.120		0.700	0.2	0.180	

*For Ant-1.5, Ant-1.6, Ivy-2.0, and Jedit-4.1, the cutoff threshold of prediction is 60%; for others, the cutoff threshold is 50%.

significant (marked in gray background). For all three indicators, the magnitude of the difference measured by Cliff's δ value is trivial as well (|δ| < 0.147).

3. Compared with the three baseline methods in the APV scenario.

As shown in Table 4, we observe that TAL achieves the best average performance across 10 cross-version pairs in terms of all three indicators in the APV scenario. The *p*-values of the Wilcoxon test indicate that the performance differences between TAL and E-only are statistically significant as well as the difference between TAL and E-prior. For the difference between TAL and E-pseudo, although the *p*-values of the Wilcoxon test are not statistically significant for three indicators (marked in gray background), the magnitude of the difference is not trivial for F1 and BPP (0.17 and 0.18).

Hence, the reason for the insignificant difference between TAL and E-pseudo could be due to the small sample size with only 10 samples in the APV scenario.

To observe the difference in prediction performance between TAL and E-pseudo in detail, we compare the prediction performance on the remaining unlabeled modules after selecting one module at each iteration of the active learning stage. In other words, the number of the iteration is identical to the number of the indicator's values for comparison from TAL and E-pseudo.

Table 5 shows the comparison results of 10 cross-version pairs in the APV scenario. In Table 5, *win* refers to the number of the indicator value of TAL that is significantly better than that of the E-pseudo method; *tie* refers to the number of the indicator value that has no significant difference; and *loss* refers to the number of the indicator value of TAL that is significantly worse than that of the E-pseudo method. For all three indicators, the *win*/*tie*/*loss* is both 6/2/2, which indicates that the prediction performance of TAL is obviously superior to the E-pseudo over the cross-version pair in the APV scenario.

Indeed, the good performance achieved by TAL at the first active learning phase is determined by the accuracy of the voting score, which in turn is obtained by the accuracy of the threshold calculation for each metric of the committee. That is, the thresholds of the metrics are continuously derived closer to the thresholds with the data features of the current version. TAL has more accurate metric thresholds than the other three baseline methods based on the mixed labeled module pool, so its prediction performance is also superior to the other three baseline methods.

To conclude, except for the E-only, TAL and its two variant methods, that is, E-pseudo and E-prior, can be applied in different scenarios in practice: E-pseudo can be used for defect prediction instead of TAL when the labeled modules of the prior version are not available; the E-prior can be applied in defect prediction in the SPV scenario; and TAL can be implemented for both SPV and APV scenarios, especially in APV scenario.

6.2 | Threats to validity

The work presented in this study was carefully planned and executed, but there exist threats to construct, internal, and external validity.

6.2.1 | Construct validity

We use three extensively used indicators to evaluate the performance of our TAL and baseline methods. As shown in Equation (5) in Section 5.2, GM is proposed to combine two evaluations: the accuracy of positives and the accuracy of negatives, because the effect of the negative cases (the clean classes) prevails in an imbalanced dataset.[14] Nevertheless, other comprehensive indicators, such as MCC and AUC, can also be examined.

6.2.2 | Internal validity

The first threat is from the selection of weak baseline models. We use two supervised methods (LR and RF) and the HALKP method as the baseline methods in RQ1. On the one hand, by a systematic review, Hall et al.[69] revealed that LR performed well compared with a lot of other modeling techniques. Tantithamthavorn et al.[52] reported that RF was one of the top learners among a large number of modeling techniques. Therefore,

TABLE 5 The comparison results between TAL and E-pseudo in the APV scenario.

F1			GM			BPP		
Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss
6	2	2	6	2	2	6	2	2

LR and RF can be considered strong baseline models. On the other hand, HALKP is the state-of-the-art active learning defect prediction model. In this sense, we believe that such a threat should not have a large influence on our conclusion. Nevertheless, other common supervised methods can also be considered in the future work. The second threat is from the labeling for the selected unlabeled module of the current version. In principle, domain experts should be asked to label the module. However, in our study, we just simply feed the candidate unlabeled modules with the actual labels instead of getting their oracles from the experts. This may not be a good representation of the real application scenario. This threat should be reduced in the future work.

6.2.3 | External validity

In our experiments, we used the public defect dataset available at the MORPH dataset. Although this dataset has been used by many other studies (Xu et al.[1]; Chen et al.[45]; Bennin et al.[46]), our results may be under threat if the dataset is seriously flawed. It is desirable to replicate the experiment on industrial for further investigations to evaluate their validity.

7 | CONCLUSION AND FUTURE WORK

We propose a TAL method that consists of two stages: At the active learning stage, a committee of investigated metrics is constructed to vote on the unlabeled modules of the current version. We pick up unlabeled modules with the median of voting scores to domain experts. The domain experts test and label the selected unlabeled modules. Then, we merge the labeled modules assigned by domain experts and the remaining modules with pseudo-labels from the current version into the labeled modules of the prior version to form an enhanced training data. After that, we calculate the threshold values of investigated metrics for the next iteration. At the defect prediction stage, the iterations stop when a predefined threshold is reached. Finally, we select a cutoff threshold on the voting score to predict defect-prone of remaining unlabeled modules.

We evaluate the TAL on 31 versions of 10 projects from a public dataset. Our conclusions are summarized as follows:

1. Our TAL method outperforms the baseline methods, including LR, RF, and the state-of-the-art method HALKP.
2. The select strategy of TAL is better than the other two strategies, i.e., TAL-max and TAL-min.
3. The cutoff threshold of 50% used in TAL has the best performance of defect prediction as well as the cutoff threshold of 60% compared to the other seven cutoff thresholds.

In conclusion, TAL can effectively address the different data distribution between successive versions, because it has the aforementioned strong predictive performance. Furthermore, TAL and its two variant methods can be applied in different scenarios in practice: E-pseudo can be used for defect prediction when the datasets of the prior version are not available; E-prior can be applied in defect prediction in the SPV scenario; and TAL can be implemented for SPV and APV scenarios especially in APV scenario. In addition, to keep the cost of extensive testing low, selecting 5% of candidate modules from the current version is sufficient for TAL to achieve good performance of defect prediction.

Our important future work involves replicating our experiments on other types of defect datasets, specifically on industrial datasets, to further evaluate their validity. In addition, more optimization techniques, for example, Hyperparameter Optimization,[70] will be explored to improve the sample selection strategy in future work.

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available in the supplementary material of this article.

ORCID

Yuanqing Mei https://orcid.org/0000-0003-3122-8887
Yuming Zhou https://orcid.org/0000-0002-4645-2526

REFERENCES

1. Xu Z, Liu J, Luo X, Zhang T. *Cross-version defect prediction via hybrid active learning with kernel principal component analysis*; 2018:209-220.
2. Lu H, Kocaguneli E, Cukic B. Defect prediction between software versions with active learning and dimensionality reduction. In: *2014 IEEE 25th International Symposium on Software Reliability Engineering*; 2014:312-322.
3. Li M, Zhang H, Wu R, Zhou Z-H. Sample-based software defect prediction with active and semi-supervised learning. *Autom Softw Eng*. 2012;19(2): 201-230. doi:10.1007/s10515-011-0092-1
4. Mei YQ, G. Z, Zhou HC, et al. Deriving object-oriented metric thresholds: research problems, Progress, and challenges. *Ruan Jian Xue Bao/J Softw (in Chinese)*. 2022;1-53.

5. Lanza M, Marinescu R. *Object-Oriented Metrics in Practice—Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Berlin Heidelberg New York: Springer; 2006.
6. Alves TL, Ypma C, Visser J. Deriving metric thresholds from benchmark data. *IEEE Int Conf Softw Maintenance*. 2010;1-10.
7. Ferreira KAM, Bigonha MAS, Bigonha RS, Mendes LFO, Almeida HC. Identifying thresholds for object-oriented software metrics. *J Syst Softw*. 2012; 85(2):244-257. doi:10.1016/j.jss.2011.05.044
8. Vale GAD, Figueiredo EML. A method to derive metric thresholds for software product lines. In: *2015 29th Brazilian Symposium on Software Engineering*; 2015:110-119.
9. Oliveira P, Valente MT, Bergel A, Serebrenik A. Validating metric thresholds with developers: an early result. In: *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE; 2015:546-550.
10. Shatnawi R, Li W, Swain J, Newman T. Finding software metrics threshold values using ROC curves. *J Softw Maintenance Evol Res Practice*. 2010;22(1): 1-16. doi:10.1002/smr.404
11. Rahman F, Posnett D, Devanbu P. Recalling the "imprecision" of cross-project defect prediction. In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*; 2012:1-11.
12. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng*. 2007;33(1):2-13. doi:10.1109/TSE.2007.256941
13. Zhou Y, Xu B, Leung H, Chen L. An in-depth study of the potentially confounding effect of class size in fault prediction. *ACM Trans Softw Eng Methodol*. 2014;23(1):10:11-10:51.
14. Shatnawi R. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *IEEE Trans Softw Eng*. 2010; 36(2):216-225. doi:10.1109/TSE.2010.9
15. Matwin MK a S. *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection*. Morgan Kaufmann; 1997:179-186.
16. Mauša G, Grbac TG. The stability of threshold values for software metrics in software defect prediction. In: *International Conference on Model and Data Engineering*. Cham: Springer International Publishing; 2017:81-95. doi:10.1007/978-3-319-66854-3_7
17. Arar OF, Ayan K. Deriving thresholds of software metrics to predict faults on open source software: replicated case studies. *Expert Syst Applic*. 2016; 61:106-121. doi:10.1016/j.eswa.2016.05.018
18. Boucher A, Badri M. Software metrics thresholds calculation techniques to predict fault-proneness: an empirical comparison. *Inf Softw Technol*. 2018; 96:38-67. doi:10.1016/j.infsof.2017.11.005
19. Padhy N, Panigrahi R, Neeraja K. Threshold estimation from software metrics by using evolutionary techniques and its proposed algorithms, models. *Evol Intell*. 2021;14(2):315-329.
20. Tong S, Koller D. Support vector machine active learning with applications to text classification. *J Mach Learn Res*. 2001;2:45-66.
21. Balcan M-F, Broder A, Zhang T. *Margin based active learning*. Berlin Heidelberg: Springer; 2007:35-50.
22. Freund Y, Seung HS, Shamir E, Tishby N. Selective sampling using the query by committee algorithm. *Mach Learn*. 1997;28(2):133-168. doi:10.1023/A:1007330508534
23. Roy N, McCallum A. Toward optimal active learning through sampling estimation of error reduction. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc.; 2001:441-448.
24. Dasgupta S, Hsu D. Hierarchical sampling for active learning. In: *Proceedings of the 25th International Conference on Machine Learning*. Helsinki, Finland: Association for Computing Machinery; 2008:208-215.
25. Nguyen HT, Smeulders A. Active learning using pre-clustering. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. Banff, Alberta, Canada: Association for Computing Machinery; 2004:79.
26. Huang SJ, Jin R, Zhou ZH. Active learning by querying informative and representative examples. *IEEE Trans Pattern Anal Mach Intell*. 2014;36(10): 1936-1949. doi:10.1109/TPAMI.2014.2307881
27. Yubin Qu FL, Chen X. LAL: meta-active learning-based software defect prediction. *Int J Performability Eng*. 2020;16(2):203-213. doi:10.23940/ijpe.20.02.p5.203213
28. Yubin Qu XC, Chen R, Ju X, Guo J. Active learning using uncertainty sampling and query-by-committee for software defect prediction. *Int J Performability Eng*. 2019;15(10):2701-2708. doi:10.23940/ijpe.19.10.p16.27012708
29. Fang Li YQ, Ji J, Zhang D, Li L. Active learning empirical research on cross-version software defect prediction datasets. *Int J Performability Eng*. 2020; 16(4):609-617.
30. Lu H, Cukic B. An adaptive approach with active learning in software fault prediction. In: *Proceedings of the 8th International Conference on Predictive Models in Software Engineering* (Lund, Sweden, 2012). Association for Computing Machinery:79-88.
31. Luo G, Ma Y, Qin K. Active learning for software defect prediction. *IEICE Trans Inf Syst*. 2012;E95-D(6):1680-1683. doi:10.1587/transinf.E95.D.1680
32. Mi W, Li Y, Wen M, Chen Y. Using active learning selection approach for cross-project software defect prediction. *Connection Science*. 2022;34(1): 1482-1499. doi:10.1080/09540091.2022.2077913
33. Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans Softw Eng*. 2018;44(9): 811-833. doi:10.1109/TSE.2017.2724538
34. Zhou Y, Yang Y, Lu H, et al. How far we have progressed in the journey? An examination of cross-project defect prediction. *ACM Trans Softw Eng Methodol*. 2018;27(1):1-51. doi:10.1145/3183339
35. Cao Q, Sun Q, Cao Q, Tan H. Software defect prediction via transfer learning based neural network. In: *Proceedings of the 1st International Conference on Reliability Systems Engineering (ICRSE'15)*; 2015:1-10.
36. Jing X, Wu F, Dong X, Qi F, Xu B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. Italy, ACM: Bergamo; 2015:496-507.
37. Thomas Zimmermann NN, Gall H, Giger E, Murphy B. 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 International Symposium on Foundations of Software Engineering (FSE'09)*; 2009:91-100.
38. Rathore SS, Kumar S. An empirical study of ensemble techniques for software fault prediction. *Appl Intell*. 2021;51(6):3615-3644. doi:10.1007/s10489-020-01935-6

39. Bal PR, Kumar S. WR-ELM: weighted regularization extreme learning machine for imbalance learning in software fault prediction. *IEEE Trans Reliabil*. 2020;69(4):1355-1375. doi:10.1109/TR.2020.2996261
40. Rathore SS, Kumar S. An approach for the prediction of number of software faults based on the dynamic selection of learning techniques. *IEEE Trans Reliabil*. 2019;68(1):216-236. doi:10.1109/TR.2018.2864206
41. Peng H, Bing L, Xiao L, Jun C, Yutao M. An empirical study on software defect prediction with a simplified metric set. *Inf Softw Technol*. 2015;59:170-190.
42. Bennin KE, Toda K, Kamei Y, Keung J, Monden A, Ubayashi N. Empirical evaluation of cross-release effort-aware defect prediction models. In: *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*; 2016.
43. Amasaki S. Cross-version defect prediction: use historical data, cross-project data, or both? *Empir Softw Eng*. 2020;25(2):1573-1595. doi:10.1007/s10664-019-09777-8
44. Turhan B. On the dataset shift problem in software engineering prediction models. *Empir Softw Eng*. 2012;17(1):62-74. doi:10.1007/s10664-011-9182-8
45. Chen L, Fang B, Shang Z, Tang Y. Negative samples reduction in cross-company software defects prediction. *Inf Softw Technol*. 2015;62:67-77. doi:10.1016/j.infsof.2015.01.014
46. Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S. MAHAKIL: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Trans Softw Eng*. 2018;44(6):534-550. doi:10.1109/TSE.2017.2731766
47. Zhang F, Mockus A, Keivanloo I, Zou Y. Towards building a universal defect prediction model with rank transformed predictors. *Empir Softw Eng*. 2016;21(5):2107-2145. doi:10.1007/s10664-015-9396-2
48. Wilcoxon F. *Individual Comparisons by Ranking Methods*. New York: Springer; 1992:196-202.
49. Arcuri A, Briand L. *A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering*; 2011:1-10.
50. Cliff N. *Ordinal Methods for Behavioral Data Analysis*. New York: Psychology Press; 2014. doi:10.4324/9781315806730
51. Romano, J. and Kromrey, J. *Appropriate Statistics for Ordinal Level Data: Should We Really Be Using t-test and Cohen's d for Evaluating Group Differences on the NSSE and other Surveys?* (2006).
52. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. The impact of automated parameter optimization on defect prediction models. *IEEE Trans Softw Eng*. 2019;45(7):683-711. doi:10.1109/TSE.2018.2794977
53. Jelihovschi E, Faria JC, Allaman IB. *The ScottKnott Clustering Algorithm*. Ilheus, Bahia, Brasil: Universidade Estadual de Santa Cruz-UESC; 2014.
54. Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Trans Softw Eng*. 2008;34(4):485-496. doi:10.1109/TSE.2008.35
55. Yang X, Lo D, Xia X, Zhang Y, Sun J. *Deep Learning for Just-in-Time Defect Prediction*; 2015:17-26.
56. Yang Y, Zhou Y, Lu H, et al. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. *IEEE Trans Softw Eng*. 2015;41(4):331-357. doi:10.1109/TSE.2014.2370048
57. Yang, Y., Harman, M., Krinke, J., Islam, S., Binkley, D., Zhou, Y. and Xu, B. *An Empirical Study on Dependence Clusters for Effort-Aware Fault-Proneness Prediction*, 2016.
58. Panichella A, Oliveto R, Lucia AD. *Cross-Project Defect Prediction Models: L'Union Fait La Force*; 2014:164-173.
59. Mockus A, Ping Z, Li PL. *Predictors of Customer Perceived Software Quality*; 2005:225-233.
60. Liu Y, Khoshgoftaar TM, Seliya N. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Trans Softw Eng*. 2010;36(6):852-864. doi:10.1109/TSE.2010.51
61. Soe YN, Santosa PI, Hartanto R. *Software Defect Prediction Using Random Forest Algorithm*; 2018:1-5.
62. Kumar KV, Kumari P, Chatterjee A, Mohapatra DP. *Software Fault Prediction Using Random Forests*. Springer Singapore; 2021:95-103.
63. Magal RK, Jacob S. Improved random Forest algorithm for software defect prediction through data mining techniques. *Int J Comput Applic*. 2015;117:18-22.
64. Ibrahim DR, Ghnemat R, Hudaib A. *Software Defect Prediction using Feature Selection and Random Forest Algorithm*; 2017:252-257.
65. Matloob F, Ghazal TM, Taleb N, et al. Software defect prediction using ensemble learning: a systematic literature review. *IEEE Access*. 2021;9:98754-98771. doi:10.1109/ACCESS.2021.3095559
66. Pushphavathi TP, Suma V, Ramaswamy V. *A Novel Method for Software Defect Prediction: Hybrid of FCM and Random Forest*; 2014:1-5.
67. Kakkar M, Jain S. *Feature Selection in Software Defect Prediction: A Comparative Study*; 2016:658-663.
68. Breiman L. Random forests. *Mach Learn*. 2001;45(1):5-32. doi:10.1023/A:1010933404324
69. Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans Softw Eng*. 2012;38(6):1276-1304. doi:10.1109/TSE.2011.103
70. Nevendra M, Singh P. Empirical investigation of hyperparameter optimization for software defect count prediction. *Expert Syst Applic*. 2022;191:116217. doi:10.1016/j.eswa.2021.116217

How to cite this article: Mei Y, Liu X, Lu Z, Yang Y, Liu H, Zhou Y. Cross-version defect prediction using threshold-based active learning. *J Softw Evol Proc*. 2024;36(4):e2563. doi:10.1002/smr.2563

APPENDIX A

Tables A.1, A.2, and A.3 provide the comparison results for TAL with LR, RF, and HALKP methods under four predefined thresholds, respectively.

TABLE A.1 The comparison results for TAL and LR under four predefined thresholds.

	F1				GM				BPP			
Threshold	TAL (Avg.)	LR (Avg.)	Wilcoxon test's *p*	Cliff's δ	TAL (Avg.)	LR (Avg.)	Wilcoxon test's *p*	Cliff's δ	TAL (Avg.)	LR (Avg.)	Wilcoxon test's *p*	Cliff's δ
5%	0.525	0.350	<0.001	0.532	0.655	0.454	<0.001	0.754	0.650	0.468	<0.001	0.798
10%	0.531	0.356	<0.001	0.523	0.656	0.460	<0.001	0.750	0.652	0.472	<0.001	0.786
15%	0.538	0.365	<0.001	0.521	0.661	0.467	<0.001	0.744	0.656	0.478	<0.001	0.773
20%	0.543	0.372	<0.001	0.519	0.664	0.470	<0.001	0.719	0.658	0.481	<0.001	0.761

TABLE A.2 The comparison results for TAL and RF under four predefined thresholds.

	F1				GM				BPP			
Threshold	TAL (Avg.)	RF (Avg.)	Wilcoxon test's *p*	Cliff's δ	TAL (Avg.)	RF (Avg.)	Wilcoxon test's *p*	Cliff's δ	TAL (Avg.)	RF (Avg.)	Wilcoxon test's *p*	Cliff's δ
5%	0.525	0.381	0.004	0.382	0.655	0.494	<0.001	0.598	0.650	0.500	<0.001	0.609
10%	0.531	0.387	0.004	0.386	0.656	0.499	<0.001	0.596	0.652	0.504	<0.001	0.611
15%	0.538	0.395	0.004	0.371	0.661	0.505	<0.001	0.598	0.656	0.508	<0.001	0.621
20%	0.543	0.403	0.004	0.375	0.664	0.509	<0.001	0.575	0.658	0.513	<0.001	0.594

TABLE A.3 The comparison results for TAL and HALKP under four predefined thresholds.

	F1				GM				BPP			
Threshold	TAL (Avg.)	HALKP (Avg.)	Wilcoxon test's *p*	Cliff's δ	TAL (Avg.)	HALKP (Avg.)	Wilcoxon test's *p*	Cliff's δ	TAL (Avg.)	HALKP (Avg.)	Wilcoxon test's *p*	Cliff's δ
5%	0.525	0.480	0.085	0.161	0.655	0.592	0.007	0.399	0.650	0.580	0.002	0.480
10%	0.531	0.497	0.131	0.103	0.656	0.603	0.011	0.361	0.652	0.589	0.004	0.428
15%	0.538	0.498	0.065	0.140	0.661	0.599	0.009	0.392	0.656	0.586	0.002	0.465
20%	0.543	0.509	0.096	0.103	0.664	0.617	0.024	0.326	0.658	0.603	0.010	0.421