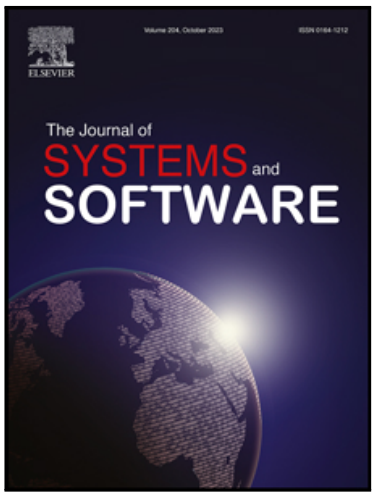


# Improve cross-project just-in-time defect prediction with dynamic transfer learning

Hongming Dai , Jianqing Xi , Hong-Liang Dai

PII: S0164-1212(24)00258-9
DOI: https://doi.org/10.1016/j.jss.2024.112214
Reference: JSS 112214

To appear in: *The Journal of Systems & Software*

Received date: 30 June 2024
Revised date: 20 August 2024
Accepted date: 8 September 2024

Please cite this article as: Hongming Dai , Jianqing Xi , Hong-Liang Dai , Improve cross-project just-in-time defect prediction with dynamic transfer learning, *The Journal of Systems & Software* (2024), doi: https://doi.org/10.1016/j.jss.2024.112214

# Hightlights

- Introduction of the kernel variance matching method to address variations in marginal probability distributions between the source and target projects
- Utilization of the CatBoost algorithm for just-in-time software defect prediction model construction
- Introduction of the improved CORAL method for formulating the model's loss function
- Introduction of the KCC method to handle various marginal and conditional probability distributions of features for cross-project just-in-time defect prediction

# Improve cross-project just-in-time defect prediction with dynamic transfer learning

Hongming Dai [a, b, *], Jianqing Xi [a, *], and Hong-Liang Dai [c]
[a]School of Software, South China University of Technology, Guangzhou 510006, China
[b]School of Information, Guangdong Polytechnic of Science and Trade, Guangzhou 510430, China
[c]School of Economics and Statistics, Guangzhou University, Guangzhou 510006, China

***Abstract***

Cross-project just-in-time software defect prediction (CP-JIT-SDP) is a prominent research topic in the field of software engineering. This approach is characterized by its immediacy, accuracy, real-time feedback, and traceability, enabling it to effectively address the challenges of defect prediction in new projects or projects with limited training data. However, CP-JIT-SDP faces significant challenges due to the differences in the feature distribution between the source and target projects. To address this issue, researchers have proposed methods for adjusting marginal or conditional probability distributions. This study introduces a transfer-learning approach that integrates dynamic distribution adaptation. The kernel variance matching (KVM) method is proposed to adjust the disparity in the marginal probability distribution by recalculating the variance of the source and target projects within the reproducing kernel Hilbert space (RKHS) to minimize the variance disparity. The categorical boosting (CatBoost) algorithm is used to construct models, while the improved CORrelation ALignment (CORAL) method is applied to develop the loss function to address the difference in the conditional probability distribution. This method is abbreviated as KCC, where the symbol K represents KVM, the symbol C represents CatBoost, and the next symbol C represents improved CORAL. The KCC method aims to optimize the joint probability distribution of the source project so that it closely agrees with that of the target project through iterative and dynamic integration. Six well-known open-source projects were used to evaluate the effectiveness of the proposed method. The empirical findings indicate that the KCC method exhibited significant improvements over the baseline methods. In particular, the KCC method demonstrated an average increase of 18% in the geometric mean (G-mean), 105.4% in the Matthews correlation coefficient (MCC), 25.6% in the F1-score, and 16.9% in the area under the receiver operating characteristic curve curve (AUC) when compared to the baseline methods. Furthermore, the KCC method demonstrated greater stability.

Index Terms
CatBoost, correlation alignment, cross-project, just-in-time defect prediction, kernel variance matching

## 1. INTRODUCTION

The cost of fixing software defects decreases with early detection. Predictive technology can identify defects in their early stages, helping developers and testers restrict the scope of investigation and optimize the allocation of testing resources [1]. Studies on software defect prediction (SDP) have been conducted for more than 40 years. The potential for predicting defects has attracted significant attention from researchers since its inception. Conventional defect prediction methods primarily focus on predicting defects at the module level to identify files, packages, or subsystems that contain defects [2, 3, 4]. However, because of the constraints of this method in precisely identifying flaws, delivering timely feedback, and promptly attributing responsibility to defect resolution, researchers have proposed the implementation of just-in-time (JIT) defect prediction technology to address these challenges [5].

Just-in-time software defect prediction (JIT-SDP) technology is used to predict whether a code change submitted by developers contains defects. A key feature is its ability to quickly analyze changes in defects and predict the probability of their presence immediately after the developer submits the code change. This technology effectively addresses the challenges faced by conventional SDP techniques, particularly in the following four areas.

1) Instantaneous: JIT-SDP technology can predict defects at the moment of code change submission. Currently, developers have a fresh memory of the changes made to the code, enabling them to bypass the need to invest time in reacquainting themselves with their modified code. This facilitates the prompt correction of the defects.

2) Precision: JIT-SDP technology focuses on the analysis of individual code changes, enabling the prediction of defects at a more detailed level. Through the analysis of individual code changes, a more accurate determination of whether the change contains defects is possible, thereby assisting developers in promptly identifying and addressing any issues.

3) Real-time feedback: JIT-SDP technology can provide developers with immediate feedback by alerting them of potential defects in the code changes they have submitted. The availability of real-time feedback assists developers in promptly identifying issues in the development process and implementing appropriate resolution measures.

4) Enhanced traceability: Code modifications made by developers contain their information, enabling project managers to easily identify the developers accountable for introducing defects. This process enables a timely analysis of the factors that contribute to

* Hongming Dai(sebilldai@mail.scut.edu.cn), Jianqing Xi (cxylab@163.com)

the emergence of defects and assists in allocating tasks for defect resolution.

Currently, there is significant interest among researchers in the field of software engineering regarding the advantages of JIT-SDP technology, resulting in its incorporation to satisfy industry requirements. The main focus of the research is on the JIT-SDP within the same project. The procedure involves utilizing a subset of the historical change dataset from the project as the training set to construct a defect prediction model. The remaining unselected change data are subsequently used as the test set to assess the predictive performance of the model. In practical scenarios, the target project for defect prediction could be recently initiated, or there may be limited existing training data for this project while requiring high annotation costs and an elevated risk of errors. To address this issue, a feasible approach involves the use of pre-existing high-quality defect change datasets from other projects to construct a defect prediction model for the specific target project. This approach is valuable because it allows for the transferability of knowledge across different projects, thereby enabling more accurate predictions of defects in new projects where limited or no historical data are available. By leveraging data from multiple projects, cross-project just-in-time software defect prediction (CP-JIT-SDP) can generate more robust and generalizable models that can be applied to new projects effectively.

However, there are challenges associated with CP-JIT-SDP that need to be addressed. One of the main challenges is the heterogeneity of data across projects, including differences in domains, software development processes, development technologies, coding conventions, and development personnel. These factors can lead to significant variations in the distribution of metrics within their change datasets. These variations relate to differences in the types and frequencies of features, such as code metrics and patterns, found in the source and target projects. These inconsistencies affect the accuracy and effectiveness of SDP models in various ways. First, if the distribution of features in the target project differs significantly from that in the source project, the model trained on the source project may fail to accurately capture the patterns and characteristics of defects in the target project. Consequently, the reliability of the model's predictions may be reduced. Second, the models are predicated on the assumption that the associations between features and defects remain consistent across various projects. However, if there is a substantial variation in the distribution of features, these relationships may cease to be applicable. For example, a code metric that exhibits a high correlation with defects in the source project may not necessarily demonstrate the same relationship with defects in the target project. This discrepancy may result in inaccurate forecasts and diminished model efficacy. This presents a challenge in meeting the assumption of feature independence and identical distribution.

Several researchers have widely applied transfer learning to address CP-JIT-SDP problems [6-10, 61]. Transfer learning is a novel learning paradigm that allows algorithms to extract knowledge from one or multiple application scenarios to improve learning performance in the target scenario [11]. Thus, it can be concluded that CP-JIT-SDP is a significant application scenario for transfer learning in the domain of SDP. The dataset for JIT-SDP includes a wide range of feature measurements and related statistical data derived from software products and development processes. The feature distribution includes both the marginal and conditional probability distributions. Various feature distributions contribute distinctly to domain divergence [12]. Current studies in this field often focus on addressing the issue of divergent feature distributions between the source and target projects. These studies usually concentrate on either the marginal probability distribution or the conditional probability distribution, with little emphasis on simultaneously addressing both distribution issues. This study proposes a new method, kernel variance matching (KVM), to address the problem of varying marginal probability distributions and the improved CORrelation ALignment (CORAL) method [13] to handle the variability in conditional probability distributions. The categorical boosting (CatBoost) algorithm [60] is used to construct the JIT-SDP model, while the improved CORAL method is used to construct the loss function of the model. This method is abbreviated as KCC, where the symbol K represents KVM, the symbol C represents CatBoost, and the next symbol C represents improved CORAL.

This study evaluated the effectiveness of the KCC method by selecting the following six well-known open-source projects: Bugzilla (BUG), Columba (COL), Jdt (JDT), Platform (PLA), Mozilla (MOZ), and Postgres (POS). These projects use Java and C/C++ technologies and have 207,697 changes, including 27,866 defect changes. The evaluation metrics used in this study were the geometric mean (G-mean), Matthews correlation coefficient (MCC), F1-score, and Area Under the Curve (AUC), which were compared with those of the baseline method. The results indicate that our approach performed better than the baseline method in all the evaluation metrics and showed greater stability.

This study makes the following contributions.

1) This study presents the KVM method as a solution to address the difference in marginal probability distributions between the source and target projects. This method differs from the classical method of kernel mean matching (KMM) [29]. This approach involves recalculating the weights of the source project to assess the discrepancy in the marginal probability distributions between the two projects. This process aims to ensure the comparability of variance in the reproducing kernel Hilbert space (RKHS) between the source and target projects. This method is the first proposal in the field of transfer learning and is designed to address the differences in marginal probability distributions between the source and target projects. This is also the first proposal to address the issue of CP-JIT-SDP.

2) This study utilizes the CatBoost algorithm to construct a JIT-SDP model and incorporates the improved CORAL method to formulate the loss function of the model. This approach aims to mitigate the disparity in conditional probability distributions between the source and target projects. The loss function calculates the square of the Frobenius norm of the covariance matrix between the source and target projects to measure the discrepancy between the two projects. This study is the first to use the

improved CORAL method to develop the loss function for the CatBoost algorithm to address the CP-JIT-SDP problem.

3) The KCC method is introduced for the first time in the context of CP-JIT-SDP to address the issue of varying marginal and conditional probability distributions of features. This approach involves iteratively merging the source and target projects using the CatBoost algorithm to optimize the joint probability distribution of the source project. This aligns it more closely with the joint probability distribution of the target project. To the best of our knowledge, this study uses a transfer-learning approach with dynamic distributional adaptation to solve the CP-JIT-SDP problem, indicating the first instance of this approach being applied in the CP-JIT-SDP domain.

The rest of this paper is organized as follows. Section 2 provides an overview of previous studies on CP-JIT-SDP. Section 3 provides a detailed explanation of the methodology employed in this study. Section 4 outlines the experimental setup and research questions addressed in this study. Section 5 presents the detailed findings and experimental results. Section 6 discusses the implications and significance of the results obtained. Section 7 examines the validity of the study. Finally, Section 8 concludes this study and offers recommendations for future studies.

## 2. Related Work

In this section, we provide a review of related studies in three areas: just-in-time defect prediction, cross-project just-in-time defect prediction, and transfer learning with dynamic distribution adaptation.

### *2.1. Just-in-Time Defect Prediction*

Mockus and Weiss [14] first proposed change-level defect prediction and utilized the change history information to construct a statistical model to forecast the likelihood of faults in new changes. They classified software change characteristics into six categories: diffusion, size, diffusion and size, interval, purpose, and experience. Their study provides an essential foundation for future studies on change-level SDP. Kamei et al. summarized the benefits of change-level SDP and enhanced the characteristics of software changes based on previous studies [5]. They divided the 14 features into five dimensions: diffusion, size, purpose, history, and experience. These characteristics were used to develop the EALR model, which was validated using six open-source datasets and five commercial datasets. The model achieved an average accuracy of 68% and an average recall of 64%. Furthermore, it successfully identified 35% of the changes that caused defects while using only 20% of the total effort. Their study also provides six open-source datasets that can serve as benchmarks for future studies in this field.

Subsequently, McIntosh et al. studied JIT models as systems evolved [15]. In a longitudinal case study of 37,524 changes in the rapidly evolving QT and OpenStack systems, they discovered that fluctuations in the characteristics of fix-inducing changes can affect the performance and interpretation of JIT models. Zhou et al. developed an SDP model called defect prediction based on deep forest (DPDF) using ensemble learning and deep learning techniques [16]. The model transforms a random forest (RF) classifier into a layered structure and utilizes a cascading strategy to identify the most important defect features. Zhao et al. proposed a simplified deep forest SDP model by modifying the state-of-the-art deep forest (SDF) model and applying it to JIT-SDP for Android mobile applications [17]. Ardimento et al. proposed a method that uses an extensive feature set comprising product and process metrics extracted from software project commits, as well as their evolution. Their approach incorporates a variation of deep temporal convolutional networks with hierarchical attention layers for defect prediction [18]. Cabral and Minku investigated the issue of conceptual drift in software change features and suggested building defect classifiers by utilizing unlabeled software changes [19].

From a project perspective, research on JIT-SDP can be divided into two categories: within-project (WP) and cross-project. The studies discussed above fall into the WP category, assuming that the feature distribution of the test set is the same as that of the training set. In contrast, the cross-project type faces challenges in prediction performance due to differences in feature distribution between the test project and the training project. To tackle the CP-JIT-SDP problem, this study uses the change datasets provided by Kamei et al. [5] and applies transfer learning with dynamic distribution adaptation (DDA).

### *2.2. Cross-Project Just-in-Time Defect Prediction*

He et al. investigated defect prediction in a cross-project context, focusing on the selection of the training data. They conducted three large-scale experiments on 34 datasets obtained from 10 open-source projects [20]. Nam et al. introduced a new approach to transfer defect learning for cross-project defect prediction by building upon transfer component analysis (TCA) [21]. This approach applies TCA and determines the appropriate normalization options based on a given pair of cross-project predictions [8]. They demonstrated that cross-project defect prediction operates in only few cases. However, training data from other projects are still valuable for constructing defect prediction models for projects without historical data if selected carefully. Kamei et al. conducted a study on cross-project models constructed by the random forest algorithm in the context of JIT prediction [22]. They found that using JIT models learned from other projects could be a feasible solution for projects with limited historical data. The researchers demonstrated that carefully selecting the data used to train these models led to improved performance in a cross-project context. Xia et al. proposed a hybrid model reconstruction approach for cross-project defect prediction, comprising two phases: a genetic algorithm phase and an ensemble learning phase [23]. These two phases produce a comprehensive set of classifiers. Several studies on effort-aware JIT-SDP have employed the EALR model as a standard benchmark approach [47–48]. These studies have

conducted comparisons of the EALR model in cross-project cross-validation scenarios.

To address the instability problem in TCA+ [21], Liu et al. introduced a two-phase transfer learning model for cross-project defect prediction [9]. Herbold et al. replicated 24 methodologies suggested by researchers from 2008 to 2015 and evaluated their effectiveness on software products from five datasets, thereby establishing a benchmark for cross-project defect prediction [24]. Zhu et al. introduced a new defect prediction model called denoising autoencoder convolutional neural network (CNN) JIT defect prediction (DAECNN-JDP), which combines a denoising autoencoder and CNN for both WP and cross-project scenarios [25]. Zheng et al. proposed a novel approach that utilizes Jensen–Shannon divergence for the automatic selection of the most similar source project to the target project for cross-project defect prediction. They further apply a grouped synthetic minority oversampling technique (SMOTE) to address class imbalance issues and employ relative density estimation to select appropriate data for the source project [51]. Lin et al. investigated the effect of data merging on the interpretation of CP-JIT-SDP models [26]. Their study examined the effectiveness of CP-JIT-SDP in practical online learning scenarios [27]. Zhang et al. proposed a feature-based ensemble modeling approach. This approach considers various project characteristics and combines models that exhibit high transferability. Pandey et al. conducted a study by investigating the utility of cross-project data and introduced a novel approach that integrates deep belief networks and long short-term memory. Wang et al. proposed a proactive deep learning-based CP-JIT-SDP approach with the objective of enhancing predictive accuracy [50].

The above studies examined the feasibility of CP-JIT-SDP from the perspective of selecting training data from source projects and constructing SDP models. In contrast, this study examines the issue of varying feature distributions between the source and target software projects. The proposed method utilizes the KVM method to account for differences in marginal probability distributions, applies the improved CORAL [13] method to address differences in conditional probability distributions, and utilizes CatBoost to construct an SDP model. To the best of our knowledge, the KVM method is first proposed in this study as a means to address the differences in marginal probability distributions between the source and target projects. Additionally, the improved CORAL method is introduced for the first time in this study to develop the loss function of the CatBoost algorithm.

### *2.3. Transfer Learning with Dynamic Distribution Adaptation*

Transfer learning is a learning framework that aims to improve performance by utilizing knowledge from one domain to another. It addresses the issue of insufficient training data in a specific domain by transferring knowledge from a different domain. Pan et al. categorized and reviewed the current progress in transfer learning for classification, regression, and clustering problems [28]. They also explored its relationship with other machine learning techniques, such as domain adaptation, multitask learning, and sample selection bias. Huang et al. proposed a nonparametric approach that generates resampling weights to address sample selection bias without requiring distribution estimation [29]. Their method matches the distributions between the training and testing sets in the feature space to address this issue. Unlike previous approaches that use training data similar to the test data, Ma et al. proposed a novel algorithm called Transfer Naive Bayes (TNB), which leverages all the relevant features in the training data. This method estimates the distribution of the test data and incorporates cross-company data information into the weights of the training data [7]. This allows for the construction of a defect prediction model using these weighted data.

Nam et al. proposed a simple, effective, and efficient method called CORAL for unsupervised domain adaptation [13]. CORAL effectively reduces domain discrepancy by aligning the second-order statistics of the source and target distributions, without requiring any labels from the target domain. Experimental results on established benchmarks confirmed the versatility of CORAL, demonstrating its applicability across various types of features, including high-performance deep features, and various tasks, such as natural language processing and computer vision. Wang et al. introduced a novel and versatile framework for cross-domain learning that leverages intra-affinity among classes to facilitate the effective transfer of intra-class knowledge [30]. Wang et al. proposed a new concept called DDA that can quantitatively evaluate the relative importance of marginal and conditional probability distributions [31]. It has been demonstrated that both the marginal probability distribution and the conditional probability distribution contribute distinctly to domain divergence. Tang et al. proposed TSboostDF, a transfer-learning algorithm that addresses knowledge transfer and class imbalance for cross-project SDP [32].

Unlike the studies mentioned above, this study proposes a novel method for domain adaptation. Inspired by the study conducted by Huang et al. [29], this study proposes the use of the KVM method to address the issue of varying marginal probability distributions between the source and target software projects. Additionally, following the study conducted by Nam et al. [13], the improved CORAL method is used to develop the loss function. The loss function measures the disparity between the two projects by computing the square of the Frobenius norm of the covariance matrix.

## 3. PROPOSED FRAMEWORK: KCC

This study provides a comprehensive description of KCC, which comprises three stages: KVM, improved CORAL, and prediction. During the KVM phase, the source project is first balanced before the KVM method is applied to calculate the weights of both the source and target projects. The weights are then used to update the source project. In the improved CORAL phase, the CatBoost algorithm is utilized to develop an SDP model. The loss function for this model is defined using the improved CORAL method. The hyperparameters of the model are tuned using a stochastic search method. After each training and optimization iteration, the improved CORAL method aligns the source project with the target project using second-order features, updates the

incoming source project, and recalculates the weights of the source and target projects using the KVM method before commencing a new training and optimization iteration. During the prediction phase, defect prediction is conducted for the target project, and the results are evaluated. The proposed KCC framework is illustrated in Fig. 1.

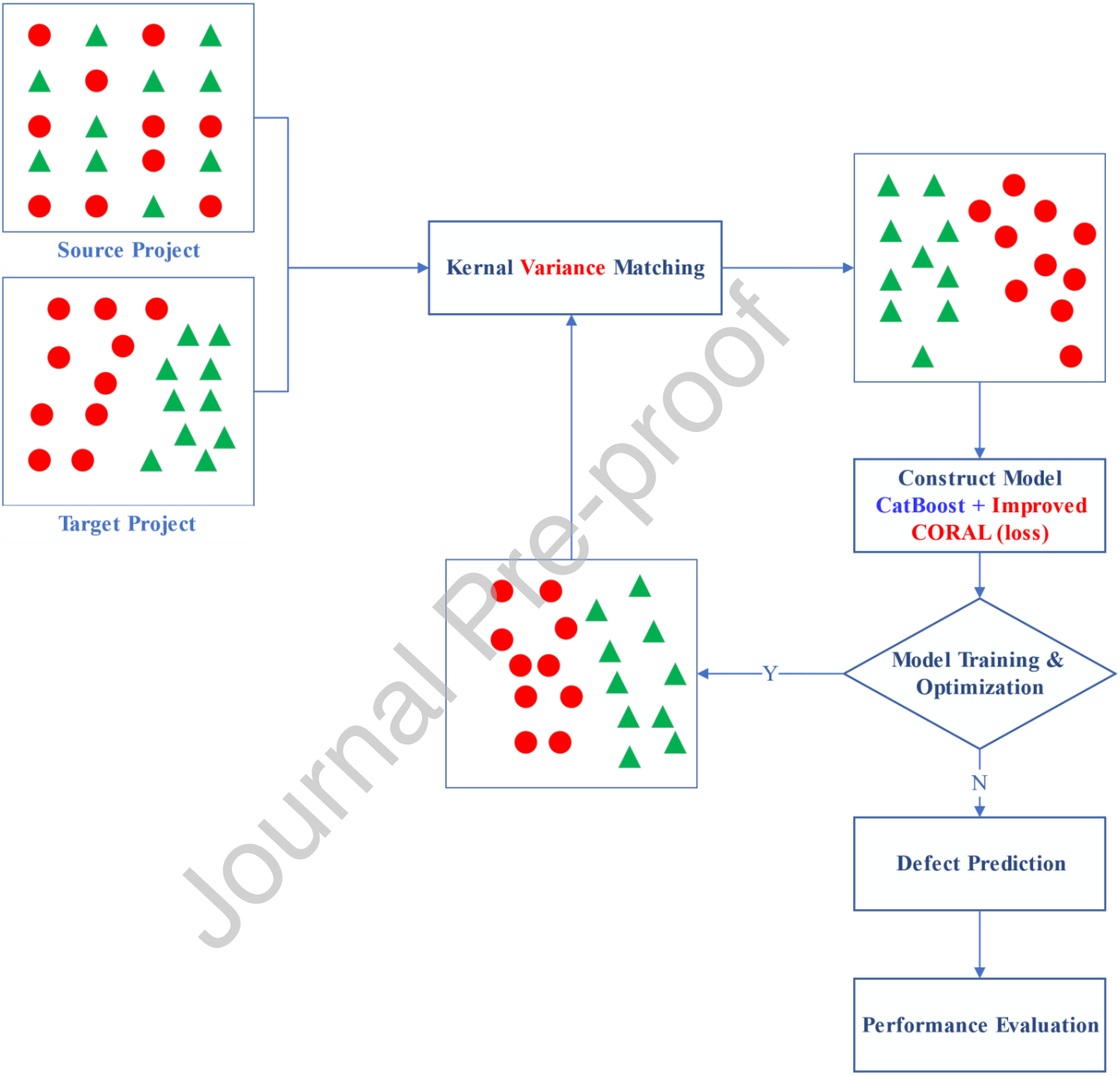


**Fig. 1.** Overall architecture diagram of the proposed framework

*3.1. Problem Description*

In this study, we aim to predict the labels for a target software project $S_t = \{x_j, y_j\}_{j=1}^{N_t}, x_j \in \chi_t, y_j \in y_t$ using labeled samples from the source software project $S_s = \{x_i, y_i\}_{i=1}^{N_s}, x_i \in \chi_{s,} y_i \in y_s$, with Ns samples to minimize prediction errors. The target project is unlabeled. The source and target software projects belong to different domains, utilize different development technologies, and are developed by different development teams. Therefore, the feature space and joint probability distribution of the two projects are not the same, but they share the same label space (0 and 1), as represented by $\chi_s \neq \chi_t, P_s(x, y) \neq P_t(x, y), y_s = y_t$. The

distribution of software project features includes marginal and conditional probability distributions. Different feature distributions contribute

differently to domain divergence [12] and can be represented as follows: (1) $P_s(x) \neq P_t(x)$, showing that the marginal difference between the two projects is distinct; and (2) $P_s(y \mid x) \neq P_t(y \mid x)$, showing that the conditional distributions of the two projects are

also dissimilar. This study aims to use a source software project to develop a prediction function ( $f : x_t \rightarrow y_t$ ) for a target software project to minimize the prediction error (measured by $\ell$ ) of the target software project, as represented below:

$$f^* = \arg\min_f \mathrm{E}_{(x,y)\in \mathrm{S}_t} \ell(f(x), y). \tag{3.1}$$

*3.2. Kernel Variance Matching*

Jing et al. proposed a comprehensive instance weighting framework for domain adaptation. The authors indicated that effective results can be achieved by incorporating and leveraging additional information from the target domain through instance weighting [33]. Huang et al. theoretically demonstrated that by adjusting the weight of the source domain, it is possible to account for the difference between $P_s(x)$ and $P_t(x)$ [29]. This is accomplished by ensuring that the means of the source and target domains in an RKHS are similar. In statistics, data distribution is related to the mean and variance. Even when the means of the data are the same, the variance and distribution of the data can differ completely. For example, consider the Gaussian distribution for $N(0,10)$, $N(0,20)$, and $N(0,30)$. Although the mean ( $\mu$ ) is 0, the variance ( $\sigma^2$ ) differs, resulting in completely different Gaussian distributions for the three types of data.

The sample mean provides information about the central position, while the variance describes not only the central position but also the shape of the distribution and the degree of dispersion. This study proposes a method called KVM to address the issue of different marginal probability distributions between the source and target software projects. Mapping the variance to the RKHS enables the analysis of the range of variation and dispersion of the samples. This approach allows for a comprehensive comparison of the differences between the two marginal probability distributions, thereby capturing changes in the data distribution across different dimensions. Consequently, it enables a more accurate and detailed evaluation of the distribution variations.

Following the approach of the classical study [33], we use maximum likelihood estimation to address the weighting issue. The parameter of the model to be learned is denoted as $\theta$, and the optimal parameter for the target software project can be expressed as follows:

$$\theta_t^* = \arg\max_\theta \int_x \sum_{y\in \mathrm{Y}} P_t(x, y) \log P(y \mid x; \theta) dx. \tag{3.2}$$

When addressing the issue of different marginal probability distributions, it is assumed that the conditional probability distributions are approximately equal, that is, $P_s(y \mid x) \approx P_t(y \mid x)$. By utilizing the Bayesian formula and empirical distributions, the model parameters of the target project can be re-expressed after transformation as follows:

$$\theta_t^* \approx \arg\max_\theta \left(\frac{1}{N_s} \sum_{i=1}^{N_s} \frac{P_t(x_i^s)}{P_s(x_i^s)} \log P(y_i^s \mid x_i^s; \theta)\right). \tag{3.3}$$

For the sake of clarity, we denote $\frac{P_t(x_i^s)}{P_s(x_i^s)}$ as the probability density ratio [29]:

$$\beta_i := \frac{P_t(x_i^s)}{P_s(x_i^s)}. \tag{3.4}$$

The optimization objective is achieved by combining the probability density ratio with the maximum mean difference (MMD) distance [34]:

$$MMD(D_s, D_t) = \sup_{f\in F} \mathrm{E}_p\left(\frac{1}{N_s} \sum_{i=1}^{N_s} \beta_i f(xi) - \frac{1}{N_t} \sum_{j=1}^{N_t} f(x_j)\right). \tag{3.5}$$

The variance of the source software project is denoted as $D_s = \mathrm{E}_{x\sim P_s(x)}\left[x - \mathrm{E}(x)\right]^2$, and the variance of the target software project is denoted as $D_t = \mathrm{E}_{x\sim P_t(x)}\left[x - \mathrm{E}(x)\right]^2$. The KVM method is represented as follows:

$$\min_{\beta} \| D_s - D_t \| = \min \| \mathrm{E}_{x \sim P_s(x)} \left[ x - \mathrm{E}(x) \right]^2 - \mathrm{E}_{x \sim P_t(x)} \left[ x - \mathrm{E}(x) \right]^2 \| . \quad (3.6)$$

After applying the kernel trick, the equation above can be simplified as follows:

$$\min_{\beta} \beta^T K \beta - 2K^T \beta \quad \text{subject to } \beta \in [0, B] \text{ and } | \sum_{i=1}^{N_s} \beta_i - N_s | \leq N_s \varepsilon , \quad (3.7)$$

where $B$ and $\varepsilon$ are the predefined thresholds. The pseudocode for the KVM method is shown in Algorithm 1.

**Algorithm 1:** Kernal Variance Matching (KVM)

**Input:** *Xs(source project dataset), Xt(target project dataset)*
**Output:** *w (the weight between Xs and Xt)*

```
// Initialization
bd = 1 // kernel bandwidth
wb = 2 // bound for w
// number of samples in source project
Ns = length(Xs)
// number of samples in target project
Nt = length(Xt)
sb = wb/∛Ns
// Compute the kernel matrix of the source project using
// the Gaussian kernel function to compute the kernel matrix
// of the source project dataset Xs
km = compute_kernel_matrix(Xs, bd)
// Using kernel functions to map the source project dataset Xs
// and the target project dataset Xt to the feature space, and
// calculating the kernel product between them
kp = compute_kernel_product(Xs, Xt, bd, Ns, Nt)
// Create a constraint matrix to limit the range of w
cm = create_constraint_matrix(Ns)
// Create a constraint vector to limit the range of w
cv = create_constraint_vector(Ns, sb, wb)
// Model the problem as a quadratic programming problem and use
// a solver to solve it, with the objective of minimizing the
// difference between the inner product of km, kp and w
solution = solve_quadratic_programming_problem(km, kp, cm, h)
// Extract the optimal w values from the solution
w = extract_optimal_w(solution)
```

### *3.3. Improved Correlation Alignment*

In examining the differences in conditional probability distributions between source and target domains, the CORAL method initiates by computing the covariance matrices of both domains [13]. It then transforms the source domain data using matrix multiplication and matrix fractional powers to account for discrepancies in conditional probability distributions between the domains. Software change datasets exhibit characteristics such as extensive scale, numerous features, and intricate structures. Despite its efficacy, the CORAL method is constrained by heightened errors and prolonged processing durations when addressing CP-JIT-SDP. To enhance the CORAL method's capacity in managing variations in conditional probability distributions for CP-JIT-SDP, improvements have been implemented. Initially, the data from the source and target domains were centralized, followed by the calculation of covariance matrices for both domains. Subsequently, the utilization of the Frobenius norm was proposed as a means of computing the loss values of the source and target domains, thereby enabling the quantification of disparities in conditional
probabilities between the two domains. The alignment of features between the source and target domains is facilitated through the loss function, thereby augmenting the model's generalization capabilities.

The improved CORAL method adjusts the input feature distributions of the source and target domains by exploring their second-order statistics. In particular, improved CORAL uses the covariance of the target distribution to recolor the source features, thereby aligning the distributions. In this study, the CatBoost algorithm is used to build the JIT-SDP, and its loss function can be customized. The improved CORAL method is used to develop the loss function for the JIT-SDP models. This loss function measures the difference between the conditional probability distributions of the source and target software projects by computing the square of the Frobenius norm of the covariance matrix. The algorithm for the loss function is presented in Algorithm 2.

**Algorithm 2: Loss Function**

**Input:** *Xs, Xt*
**Output:** *loss*

```
// dimension of samples in source project
dim = dimension(Xs)
// number of samples in source project
Ns = length(Xs)
// number of samples in target project
Nt = length(Xt)
// Centralize the source project and calculate the covariance matrix
Xs = calculate_covariance_matrix(Xs, Ns)
// Centralize the target project and calculate the covariance matrix
Xt = calculate_covariance_matrix(Xt, Nt)
// Calculate the Frobenius norm between the two projects
loss = calculate_frobenius norm(Xs − Xt, dim)
```

### *3.4. CatBoost Algorithm*

CatBoost is a gradient boosting algorithm that is both powerful and efficient, particularly well-suited for datasets that include categorical features. Its distinctive methodologies address prevalent challenges in machine learning, such as overfitting and the necessity for feature preprocessing, while simultaneously delivering optimal performance. The specific configuration of the CatBoost algorithm may include a number of hyperparameters, including the learning rate, tree depth, regularization parameters, and the number of iterations. These hyperparameters are configured with the objective of optimizing the JIT-SDP model's performance in capturing the intricate relationships within the data.

In the KCC method, the CatBoost algorithm plays a crucial role in learning the conditional probability distributions of the source and target projects. This is essential for effectively transferring sample weight from the target project to the source project, enabling the JIT-SDP model to understand the relationships between input features and the output variable in both projects. Additionally, the loss function utilized by the CatBoost algorithm is formulated using the improved CORAL method. By integrating the improved CORAL method into the loss function, the CatBoost algorithm can efficiently handle the varying conditional probability distributions between the source and target projects, thereby reducing the distribution disparities between them. This enhancement ultimately improves the transfer learning process in the KCC method.

## 4. EXPERIMENT SETUP

The objective of this study is to validate the effectiveness of the KCC method. This involves creating software change datasets for both source and target projects, creating JIT-SDP models, training these models, and assessing their performance. The experimental process follows an established scientific research methodology and employs appropriate data analysis and statistical methods. The experimental sessions evaluate the effectiveness of the KCC method in CP-JIT-SDP.

### *4.1. Studied Projects and Datasets*

This study analyzed six well-known open-source projects (BUG, COL, JDT, PLA, MOZ, and POS) that had 207,697 changes, with 27,866 of them being defect changes. These six projects cover diverse domains, developers, and development technologies, making them suitable for the CP-JIT-SDP research. Change metrics comprise process and product metrics, with each change sample containing 14 features distributed across five dimensions. Please refer to Table 1 for additional information.

First, changes are extracted from the software configuration management (SCM) system of each project. Subsequently, the SZZ algorithm and its variants [35, 36] are used to identify whether the changes introduce defects. If a defect is introduced, the change is labeled defective; otherwise, it is labeled nondefective. After constructing the defective change dataset, the statistical data for the six projects are presented in Table 2.

**Table 1** Summary of Measures for Change [5]

| Dimension | Metric | Description |
|---|---|---|
| Diffusion | NS | Modified subsystem count |
| | ND | Modified directory count |
| | NF | Modified file count |
| | Entropy | Distribution of modified code in each file |
| Size | LA | Lines of code (LOC) added |
| | LD | LOC deleted |
| | LT | Number of LOC in the file before the change |
| Purpose | FIX | Whether the change is a defect fix or not |

| | | |
|---|---|---|
| History | NDEV | The number of developers who modified the files |
| | AGE | The time elapsed between the most recent change and the current one |
| | NUC | The number of unique changes to the modified files |
| Experience | EXP | Developer experience |
| | REXP | Recent developer experience |
| | SEXP | Subsystem developer experience |

**Table 2** Statistics of the Studied Projects

| Project | Period | Clean Number | Bug Number | Average LOC /Change | Modified Files/Change |
|---|---|---|---|---|---|
| BUG | 08/1998–12/2006 | 2924 | 1696 | 591.38 | 2.29 |
| COL | 11/2002–07/2006 | 3094 | 1361 | 114.24 | 6.2 |
| JDT | 05/2001–12/2007 | 30297 | 5089 | 437.72 | 3.87 |
| PLA | 05/2001–12/2007 | 54798 | 9452 | 354.7 | 3.76 |
| MOZ | 01/2000–12/2006 | 93126 | 5149 | 970.59 | 3.7 |
| POS | 07/1996–05/2010 | 15312 | 5119 | 853.34 | 4.46 |

*4.2. Data Preprocessing*

To improve the quality of the datasets, we applied three data preprocessing techniques to all of them. First, we conducted data cleaning to identify and correct errors, missing values, outliers, and duplicates in the datasets. We addressed these issues using deletion methods.

Second, we normalized the data. The objective of data normalization is to address the issue of varying magnitudes in the values of the 14 basic change features. The min–max method is employed for this normalization process, which scales all values to fall within the range of [0, 1]. For a given feature F, its maximum and minimum values are denoted as $F_{max}$ and $F_{min}$, respectively. The normalized value $V_i$ for each value $F_i$ of feature $F$ is calculated as follows:

$$V_i = \frac{F_i - F_{min}}{F_{max} - F_{min}}.$$

Furthermore, datasets of software changes often contain imbalanced data with fewer defect-inducing changes than clean changes. This imbalance can negatively affect the performance of the model. To tackle this issue, we used undersampling techniques to randomly eliminate a specific number of clean changes. This method helps balance the number of clean changes with those that may induce defects.

*4.3. Compared Approaches*

In the field of SDP, researchers commonly use machine learning algorithms to construct SDP models. Owing to the association between SDP outcomes and variables, such as feature selection, data quality, data quantity, and algorithm selection, as well as the inherent disparities among various software projects (including diverse domains, development technologies, and developers), establishing a standardized comparison criterion is challenging. Several studies have shown that logistic regression [5, 37], RF [22, 38], and support vector machines [39] are commonly used and demonstrate superior performance despite variations in datasets and methodologies across studies.

However, the performance of these algorithms may vary based on the performance evaluation metrics used. To evaluate the performance of the KCC method, we compared it with those of four other methods. The initial baseline method is eXtreme Gradient Boosting (XGBoost) [56]. The second baseline method employs double random forests to construct SDP models [57–58], which is denoted as "DRF". The third approach is attention-based Long Short-Term Memory (LSTM) [59], which is denoted as "Attention HSTM". The last approach is the direct utilization of the CatBoost algorithm [60] to construct an SDP model, referred to as "CatBoost" .

*4.4. Classification Mode & Hyperparameter Optimization*

In this study, we use the CatBoost algorithm to build a JIT-SDP model. CatBoost is a gradient boosting algorithm known for its ability to effectively handle categorical features, which are prevalent in SDP tasks. By leveraging the powerful ensemble learning technique of CatBoost, our objective is to improve the performance of JIT-SDP. The capacity of the algorithm to handle high-dimensional data, address missing values, and provide automatic feature selection makes it an excellent choice for developing a robust JIT-SDP model.

To enhance the performance of our JIT-SDP model, we must conduct model hyperparameter optimization to improve its effectiveness. Hyperparameter optimization involves systematically selecting the best combination of hyperparameters to maximize the performance metrics of the model. In this study, we use a random search to explore the hyperparameter space of the KCC and baseline methods [54]. Random search is similar to grid search, but it differs because it does not evaluate all possible values within the search space [55]. Instead, random search selects a specified number of samples randomly within the upper and lower bounds as

potential hyperparameter values. These selected candidates are then trained until the allocated budget is utilized. One key benefit of random search is its ease of parallelization and resource allocation, as each evaluation can be conducted independently.

The principal procedures of model hyperparameter optimization are outlined as follows. The initial step is to define the hyperparameter configuration space for KCC and the baseline methods, as illustrated in Table 3. To simultaneously optimize the four evaluation metrics—G-mean, MCC, F1-score, and AUC, the optimization target of the random search was set as F1-score. The F1-score is defined as the harmonic mean of precision and recall. By employing F1-score as the optimization target, it is possible to identify hyperparameter combinations that demonstrate favorable performance across multiple evaluation metrics. A total of 200 random search iterations were conducted. Subsequently, the hyperparameter optimization process commences in an iterative looping structure.

In each iteration, CatBoost and the benchmark algorithm construct a JIT-SDP model based on the specified hyperparameters and perform model training. During the training process, the KVM method is used to adjust the marginal probability distribution of the source project to align with that of the target project. Subsequently, the improved CORAL method is employed to adjust the conditional probability distribution of the source project to match that of the target project. Evaluation metrics are then calculated. Throughout the iterations, the system keeps a record of the best solution found so far. The search process ends when a stopping criterion is met, which could involve reaching a predefined number of iterations or finding an acceptable solution. Ultimately, the random search algorithm outputs the identified optimal solution.

*4.5. Evaluation Metrics*

The primary objective of JIT-SDP is to aid software maintenance staff in precisely identifying defects and maximizing defect detection, thereby enhancing the effectiveness of SQA efforts. These requirements place constraints on the precision and recall performance of the SDP models. JIT-SDP is concerned with a binary classification task, which is often characterized by a low proportion of defect changes, resulting in an imbalanced dataset. The study uses the G-mean, MCC, F1-score, and AUC as performance evaluation metrics.

The predicted results of the model are compared with the actual label values to generate a confusion matrix, as shown in Table 4.

$$G-mean=\sqrt{Recall*Specificity}=\sqrt{\frac{TP}{TP+FN}*\frac{TN}{TN+FP}}. \tag{4.1}$$

The software change dataset shows a notable disparity between the number of defect changes and nondefect changes. In this scenario, commonly used metrics, such as accuracy, may produce misleading results. This is because the model tends to predict a higher volume of nondefect changes, resulting in increased prediction accuracy but potentially lacking in its ability to effectively identify the minority class of defect changes. The G-mean metric calculates the geometric mean of changes in defects and nondefects in the model by considering the predictive performance of both positive and negative samples [40]. This metric is particularly useful for addressing issues related to JIT-SDP:

$$MCC=\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}. \tag{4.2}$$

The MCC quantifies the correlation between the predicted and actual results of a defect prediction model, considering true positives, true negatives, false positives, and false negatives. It is suitable for evaluating the overall performance of the model when handling imbalanced samples [41].

$$F1-score=2*\frac{Precision*Recall}{Precision+Recall}. \tag{4.3}$$

The F1-score is a metric used to evaluate the performance of a model, calculated as the harmonic mean of precision and recall [52]. It can be utilized to evaluate a model's capacity to achieve equilibrium among various classes. When evaluating the prediction results of a model considering both accuracy and completeness, a higher F1-score signifies superior overall performance.

The receiver operating characteristic curve (ROC) is a method for evaluating model performance [53]. It calculates the ratio of correctly predicted defective changes to incorrectly predicted normal changes. If the data distribution is imbalanced, the performance of the SDP model can be evaluated based on the costs of misclassifying defective and normal changes. The area under the ROC (AUC) is used to measure the predictive ability of the model. An AUC value of 1 indicates that the model has 100% predictive ability, whereas an AUC value below 0 indicates that the model has no predictive ability.

The G-mean measures the prediction accuracy, while the MCC evaluates the quality of the predicted results. F1-score and AUC are employed to evaluate the quality of the model from different perspectives. Together, these metrics provide a more comprehensive evaluation of the performance of the model.

**Table 3** Configuration Space for KCC and the baseline methods

| Method | Name | Type | Range |
|---|---|---|---|
| KCC, CatBoost | learning_rate | Continuous | [0.01, 0.3] |
| | depth | Discrete | [5,20] |

| | | | |
|---|---|---|---|
| | iterations | Discrete | [100,1000] |
| | subsample | Discrete | [0.5, 1] |
| | random_strength | Continuous | [0.1, 1] |
| DRF | n_estimators | Discrete | [10,300] |
| | max_depth | Discrete | [5,20] |
| | max_features | Discrete | [1,14] |
| | min_samples_split | Discrete | [2,11] |
| | min_samples_leaf | Discrete | [1,11] |
| | criterion | Categorical | ['squared_error', 'absolute_error'] |
| Attention HSTM | learning_rate | Continuous | [0.01, 0.3] |
| | gradient clipping | Discrete | [0.5,1] |
| | batch size | Discrete | [32, 256] |
| | dropout rate | Continuous | [0.1, 0.5] |
| XGBoost | n_estimators | Discrete | [10,300] |
| | max_depth | Discrete | [5,20] |
| | learning_rate | Continuous | [0.01,0.3] |
| | subsample | Discrete | [0.5,1] |
| | colsample_bytree | Discrete | [0.5,1] |
| | gamma | Discrete | [0,10] |

**Table 4** Confusion Matrix

| | Predictive Positive | Predictive Negative |
|---|---|---|
| Real Positive | True Positive (TP) | False Negative (FN) |
| Real Negative | False Positive (FP) | True Negative (TN) |

*4.6. Statistical Tests*

To evaluate the performance disparities between the proposed method and the compared methods regarding JIT-SDP, we utilized the Scott–Knott test [42] to analyze the experimental results. The test ranks models based on their predictive performance and then utilizes hierarchical clustering analysis to categorize models with significant differences into distinct groups.

## 5. EXPERIMENTAL RESULTS

This section presents the experimental results of our method and addresses the following three questions.

**RQ1: Does our KCC method perform better than the baseline methods in terms of the G-mean?**

Because the JIT-SDP model in this study is constructed using machine learning algorithms, its prediction results are probabilistic. To ensure reliable comparative results, each method was repeated 10 times, and the average of these runs was compared. Table 5 presents the comparison results between the KCC and baseline methods for the G-mean metric, with the best values in each row highlighted in bold.

**Table 5** KCC vs. Baseline Methods (G-mean)

| Source | Target | XGBoost | DRF | Attention HSTM | CatBoost | KCC |
|---|---|---|---|---|---|---|
| **BUG** | **COL** | 0.551 | 0.551 | 0.508 | 0.606 | **0.689** |
| | **JDT** | 0.556 | 0.561 | 0.516 | 0.577 | **0.818** |
| | **MOZ** | 0.581 | 0.625 | 0.526 | 0.626 | **0.774** |
| | **PLA** | 0.666 | 0.665 | 0.647 | 0.684 | **0.747** |
| | **POS** | 0.518 | 0.483 | 0.456 | 0.556 | **0.788** |
| **COL** | **BUG** | 0.615 | 0.602 | 0.681 | 0.529 | **0.802** |
| | **JDT** | 0.605 | 0.647 | 0.661 | 0.617 | **0.738** |
| | **MOZ** | 0.617 | 0.652 | 0.636 | 0.623 | **0.717** |
| | **PLA** | 0.692 | 0.733 | 0.751 | 0.717 | **0.799** |
| | **POS** | 0.616 | 0.667 | 0.654 | 0.643 | **0.821** |
| **JDT** | **BUG** | 0.669 | 0.696 | 0.684 | 0.705 | **0.741** |
| | **COL** | 0.671 | 0.727 | 0.718 | 0.688 | **0.758** |
| | **MOZ** | 0.682 | 0.739 | 0.716 | 0.703 | **0.862** |
| | **PLA** | 0.668 | 0.741 | 0.678 | 0.705 | **0.812** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | **POS** | 0.706 | 0.725 | 0.741 | 0.721 | **0.848** |
| **MOZ** | **BUG** | 0.666 | 0.694 | 0.701 | 0.701 | **0.827** |
| | **COL** | 0.671 | 0.703 | 0.682 | 0.681 | **0.803** |
| | **JDT** | 0.681 | 0.696 | 0.694 | 0.693 | **0.759** |
| | **PLA** | 0.709 | 0.746 | 0.741 | 0.735 | **0.791** |
| | **POS** | 0.702 | 0.752 | 0.733 | 0.707 | **0.835** |
| **PLA** | **BUG** | 0.667 | 0.691 | 0.691 | 0.644 | **0.745** |
| | **COL** | 0.634 | 0.678 | 0.687 | 0.701 | **0.747** |
| | **JDT** | 0.647 | 0.651 | 0.669 | 0.657 | **0.859** |
| | **MOZ** | 0.658 | 0.702 | 0.671 | 0.669 | **0.769** |
| | **POS** | 0.672 | 0.716 | 0.664 | 0.709 | **0.801** |
| **POS** | **BUG** | 0.525 | 0.672 | 0.633 | 0.634 | **0.775** |
| | **COL** | 0.644 | 0.694 | 0.655 | 0.659 | **0.782** |
| | **JDT** | 0.666 | 0.692 | 0.687 | 0.675 | **0.789** |
| | **MOZ** | 0.676 | 0.719 | 0.704 | 0.682 | **0.786** |
| | **PLA** | 0.704 | 0.755 | 0.751 | 0.738 | **0.803** |

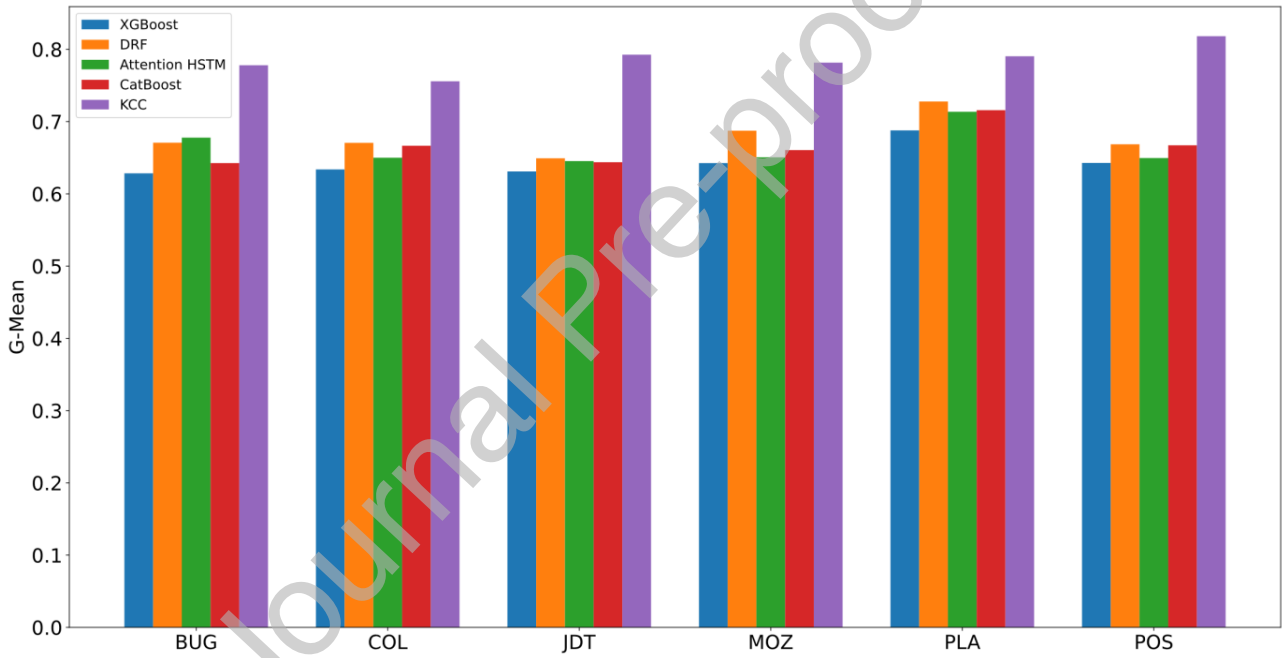


**Fig. 2.** Comparison of the average G-means of the target projects between the KCC and baseline methods

Table 5 shows that the KCC method outperformed the Attention HSTM and CatBoost methods in every iteration, with an average exceeding those of the XGBoost, DRF, Attention HSTM, and CatBoost methods by 21.5%, 15.3%, 17.8%, and 17.5%, respectively. However, the final outcomes varied when the source or target projects were altered. To evaluate the transfer effect of the KCC method, we compared the average G-mean of the target project. The comparative results are presented in Fig. 2.

Fig. 2 shows the correlation between the G-mean and different methods and projects. Out of the six projects analyzed, the KCC method displayed the most favorable performance, whereas the XGBoost method showed the least favorable performance. However, the DRF, Attention HSTM, and CatBoost methods demonstrated performance attributes comparable to each other. Upon further investigation, the average G-means for the XGBoost, DRF, Attention HSTM, CatBoost, and KCC methods have ranges of ±0.065, ±0.079, ±0.068, ±0.073, and ±0.062, respectively, across the six target projects. All the five methods demonstrate consistent performance in terms of the G-mean, while the KCC method shows the most favorable results.

**RQ2: Does our KCC method perform better than the baseline methods in terms of MCC?**

Table 6 presents the comparison results of the KCC and baseline methods using the MCC metric, with the best values in each row highlighted in bold. Table 6 shows that the KCC method outperformed the other methods in every run. However, the results varied when the source or target projects were changed. To evaluate the transfer effect of the KCC method, we compared the average

values obtained for the target projects using the MCC metric. The results are presented in Fig. 3. Fig. 3 depicts the correlation of MCC metrics with various methods and projects. Among the six projects examined, the KCC method demonstrated the most favorable performance, whereas the XGBoost method exhibited the least favorable performance. However, the DRF, Attention HSTM, and CatBoost methods displayed comparable levels of performance. Upon further analysis, the average MCC values for the XGBoost, DRF, Attention HSTM, CatBoost, and KCC methods in the six target projects range from ±0.132 to ±0.188. The results indicate that all the three methods perform similarly, as measured using the MCC metric. However, the KCC method exhibits the highest performance.

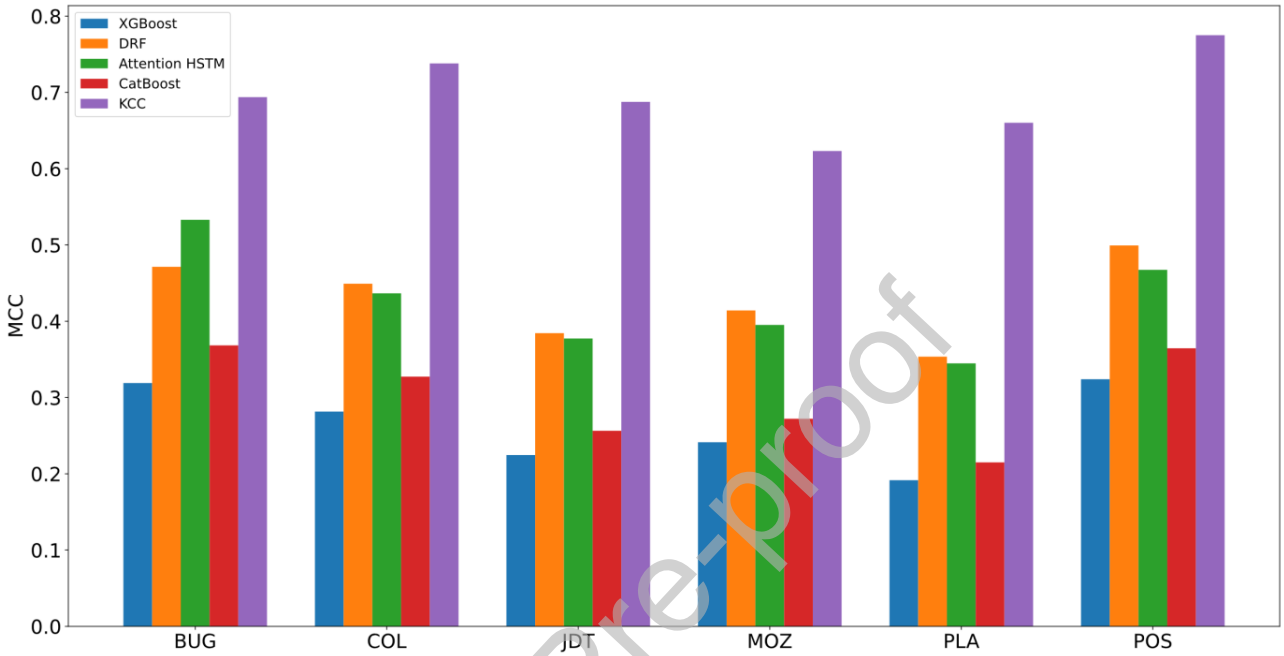


**Fig. 3.** Comparison of the average MCC of the target projects between the KCC and baseline methods

**Table 6** KCC vs. Baseline Methods (MCC)

| Source | Target | XGBoost | DRF | Attention HSTM | CatBoost | KCC |
|---|---|---|---|---|---|---|
| **BUG** | **COL** | 0.192 | 0.299 | 0.287 | 0.245 | **0.578** |
| | **JDT** | 0.166 | 0.334 | 0.336 | 0.204 | **0.638** |
| | **MOZ** | 0.186 | 0.295 | 0.324 | 0.241 | **0.651** |
| | **PLA** | 0.172 | 0.334 | 0.287 | 0.185 | **0.687** |
| | **POS** | 0.194 | 0.312 | 0.332 | 0.244 | **0.745** |
| **COL** | **BUG** | 0.315 | 0.386 | 0.526 | 0.279 | **0.724** |
| | **JDT** | 0.176 | 0.379 | 0.375 | 0.252 | **0.686** |
| | **MOZ** | 0.211 | 0.431 | 0.363 | 0.256 | **0.608** |
| | **PLA** | 0.203 | 0.389 | 0.407 | 0.243 | **0.616** |
| | **POS** | 0.315 | 0.533 | 0.463 | 0.394 | **0.775** |
| **JDT** | **BUG** | 0.327 | 0.501 | 0.486 | 0.402 | **0.599** |
| | **COL** | 0.316 | 0.531 | 0.523 | 0.354 | **0.766** |
| | **MOZ** | 0.266 | 0.469 | 0.405 | 0.297 | **0.669** |
| | **PLA** | 0.171 | 0.327 | 0.321 | 0.195 | **0.689** |
| | **POS** | 0.364 | 0.547 | 0.521 | 0.395 | **0.794** |
| **MOZ** | **BUG** | 0.322 | 0.514 | 0.582 | 0.415 | **0.785** |
| | **COL** | 0.316 | 0.479 | 0.434 | 0.343 | **0.795** |
| | **JDT** | 0.272 | 0.428 | 0.401 | 0.291 | **0.672** |
| | **PLA** | 0.196 | 0.345 | 0.308 | 0.222 | **0.658** |
| | **POS** | 0.363 | 0.522 | 0.531 | 0.365 | **0.858** |
| **PLA** | **BUG** | 0.38 | 0.491 | 0.548 | 0.371 | **0.623** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | **COL** | 0.301 | 0.453 | 0.496 | 0.383 | **0.779** |
| | **JDT** | 0.259 | 0.409 | 0.407 | 0.278 | **0.763** |
| | **MOZ** | 0.276 | 0.436 | 0.439 | 0.297 | **0.581** |
| | **POS** | 0.384 | 0.583 | 0.491 | 0.425 | **0.703** |
| **POS** | **BUG** | 0.251 | 0.467 | 0.523 | 0.375 | **0.738** |
| | **COL** | 0.283 | 0.484 | 0.443 | 0.312 | **0.772** |
| | **JDT** | 0.251 | 0.372 | 0.369 | 0.257 | **0.679** |
| | **MOZ** | 0.268 | 0.441 | 0.445 | 0.271 | **0.608** |
| | **PLA** | 0.216 | 0.373 | 0.401 | 0.231 | **0.651** |

**RQ3: Does our KCC method perform better than the baseline methods in terms of F1-score?**

Table 7 displays the comparative outcomes between the KCC and baseline methodologies using the F1-score measure, with the superior values in each row emphasized in bold. Table 7 demonstrates that the KCC method exhibited superior performance compared to the other methods across all iterations. Nevertheless, the outcomes differed with alterations in either the source or target projects. To assess the transferability of the KCC method, we contrasted the mean values achieved for the target projects by utilizing the F1-score metric. These findings are illustrated in Fig. 4.

Fig. 4 illustrates the correlation between F1-score metrics and various methods and projects. Among the six projects evaluated, the KCC method showed the most favorable performance, whereas the XGBoost method demonstrated the least favorable performance. However, the DRF, Attention HSTM, and CatBoost methods exhibited similar levels of performance. Upon closer investigation, the average F1-score values for the XGBoost, DRF, Attention HSTM, CatBoost, and KCC methods across the six target projects ranged from ±0.319 to ±0.365. These findings suggest that the three methods perform comparably based on the F1-score metric, with the KCC method exhibiting the highest performance.

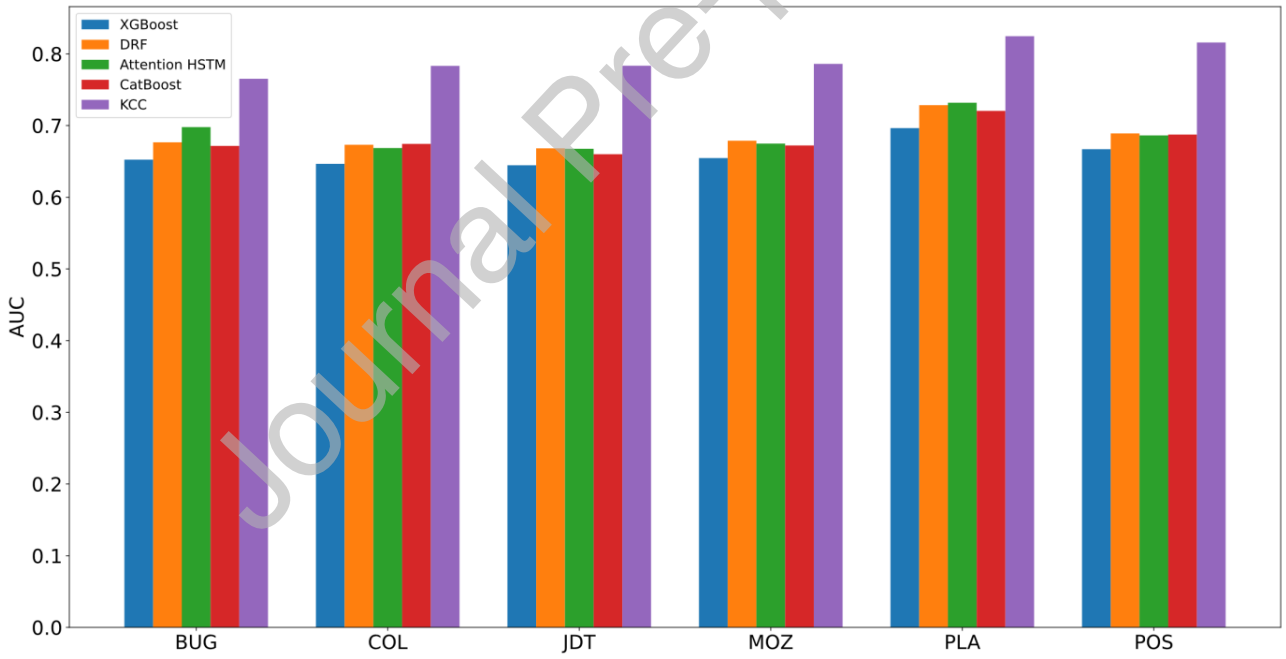


**Fig. 4.** Comparison of the average F1-score of the target projects between the KCC and baseline methods

**Table 7**
KCC vs. Baseline Methods (F1-score)

| Source | Target | XGBoost | DRF | Attention HSTM | CatBoost | KCC |
|---|---|---|---|---|---|---|
| **BUG** | **COL** | 0.413 | 0.468 | 0.457 | 0.477 | **0.534** |
| | **JDT** | 0.306 | 0.392 | 0.333 | 0.334 | **0.475** |
| | **MOZ** | 0.327 | 0.381 | 0.348 | 0.371 | **0.527** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | **PLA** | 0.194 | 0.258 | 0.261 | 0.201 | **0.388** |
| | **POS** | 0.361 | 0.371 | 0.359 | 0.405 | **0.599** |
| **COL** | **BUG** | 0.521 | 0.585 | 0.667 | 0.421 | **0.669** |
| | **JDT** | 0.322 | 0.463 | 0.429 | 0.374 | **0.544** |
| | **MOZ** | 0.351 | 0.426 | 0.451 | 0.381 | **0.467** |
| | **PLA** | 0.216 | 0.298 | 0.264 | 0.252 | **0.431** |
| | **POS** | 0.474 | 0.584 | 0.568 | 0.522 | **0.629** |
| **JDT** | **BUG** | 0.601 | 0.684 | 0.691 | 0.634 | **0.746** |
| | **COL** | 0.554 | 0.631 | 0.671 | 0.581 | **0.695** |
| | **MOZ** | 0.386 | 0.451 | 0.485 | 0.405 | **0.529** |
| | **PLA** | 0.163 | 0.269 | 0.252 | 0.182 | **0.393** |
| | **POS** | 0.546 | 0.647 | 0.634 | 0.565 | **0.665** |
| **MOZ** | **BUG** | 0.595 | 0.678 | 0.701 | 0.627 | **0.761** |
| | **COL** | 0.556 | 0.631 | 0.646 | 0.576 | **0.664** |
| | **JDT** | 0.391 | 0.489 | 0.483 | 0.404 | **0.515** |
| | **PLA** | 0.186 | 0.269 | 0.271 | 0.209 | **0.375** |
| | **POS** | 0.544 | 0.642 | 0.611 | 0.546 | **0.669** |
| **PLA** | **BUG** | 0.586 | 0.617 | 0.657 | 0.559 | **0.705** |
| | **COL** | 0.512 | 0.618 | 0.616 | 0.588 | **0.755** |
| | **JDT** | 0.383 | 0.431 | 0.478 | 0.397 | **0.559** |
| | **MOZ** | 0.399 | 0.493 | 0.459 | 0.415 | **0.498** |
| | **POS** | 0.537 | 0.635 | 0.601 | 0.575 | **0.687** |
| **POS** | **BUG** | 0.413 | 0.629 | 0.583 | 0.549 | **0.779** |
| | **COL** | 0.545 | 0.634 | 0.615 | 0.561 | **0.691** |
| | **JDT** | 0.374 | 0.463 | 0.426 | 0.378 | **0.478** |
| | **MOZ** | 0.391 | 0.442 | 0.478 | 0.391 | **0.497** |
| | **PLA** | 0.226 | 0.276 | 0.321 | 0.221 | **0.469** |

**RQ4: Does our KCC method perform better than the baseline methods in terms of AUC?**

Table 8 displays the comparative results between the KCC and baseline methods using the AUC metric, with the superior values in each row emphasized in bold. Table 8 demonstrates that the KCC method consistently outperformed the other methods in each run. However, the results varied when the source or target projects were changed. To evaluate the transfer effect of the KCC method, we compared the average values obtained for the target projects using the AUC metric. The results are presented in Fig. 5.

Fig. 5 illustrates the correlation between AUC metrics and various methods across multiple projects. In the analysis of six projects, the KCC method demonstrated the most favorable performance, while the XGBoost method showed the least favorable results. Conversely, the DRF, Attention HSTM, and CatBoost methods exhibited similar levels of performance. Further investigation revealed that the average AUC values for the XGBoost, DRF, Attention HSTM, CatBoost, and KCC methods in the six projects ranged from ±0.052 to ±0.064. These findings suggest that the three methods perform comparably based on the AUC metric, with the KCC method exhibiting the highest performance.

**Table 8**
KCC vs. Baseline Methods (AUC)

| Source | Target | XGBoost | DRF | Attention HSTM | CatBoost | KCC |
|---|---|---|---|---|---|---|
| **BUG** | **COL** | 0.591 | 0.581 | 0.593 | 0.623 | **0.678** |
| | **JDT** | 0.597 | 0.605 | 0.605 | 0.616 | **0.766** |
| | **MOZ** | 0.611 | 0.627 | 0.588 | 0.645 | **0.755** |
| | **PLA** | 0.671 | 0.685 | 0.679 | 0.686 | **0.814** |
| | **POS** | 0.586 | 0.566 | 0.581 | 0.611 | **0.822** |
| **COL** | **BUG** | 0.645 | 0.632 | 0.692 | 0.609 | **0.697** |
| | **JDT** | 0.616 | 0.677 | 0.662 | 0.645 | **0.736** |
| | **MOZ** | 0.632 | 0.671 | 0.669 | 0.649 | **0.793** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | **PLA** | 0.697 | 0.722 | 0.749 | 0.723 | **0.812** |
| | **POS** | 0.651 | 0.695 | 0.683 | 0.679 | **0.843** |
| **JDT** | **BUG** | 0.669 | 0.689 | 0.694 | 0.705 | **0.713** |
| | **COL** | 0.671 | 0.696 | 0.721 | 0.692 | **0.762** |
| | **MOZ** | 0.682 | 0.701 | 0.717 | 0.704 | **0.792** |
| | **PLA** | 0.691 | 0.747 | 0.719 | 0.717 | **0.831** |
| | **POS** | 0.706 | 0.727 | 0.732 | 0.721 | **0.871** |
| **MOZ** | **BUG** | 0.666 | 0.679 | 0.716 | 0.706 | **0.784** |
| | **COL** | 0.671 | 0.707 | 0.674 | 0.686 | **0.835** |
| | **JDT** | 0.682 | 0.706 | 0.714 | 0.694 | **0.843** |
| | **PLA** | 0.716 | 0.731 | 0.754 | 0.738 | **0.817** |
| | **POS** | 0.702 | 0.739 | 0.731 | 0.707 | **0.742** |
| **PLA** | **BUG** | 0.682 | 0.688 | 0.702 | 0.671 | **0.811** |
| | **COL** | 0.649 | 0.683 | 0.693 | 0.702 | **0.835** |
| | **JDT** | 0.661 | 0.674 | 0.676 | 0.671 | **0.764** |
| | **MOZ** | 0.671 | 0.695 | 0.685 | 0.681 | **0.755** |
| | **POS** | 0.691 | 0.718 | 0.704 | 0.719 | **0.801** |
| **POS** | **BUG** | 0.601 | 0.695 | 0.685 | 0.668 | **0.861** |
| | **COL** | 0.653 | 0.699 | 0.663 | 0.669 | **0.806** |
| | **JDT** | 0.668 | 0.679 | 0.681 | 0.675 | **0.808** |
| | **MOZ** | 0.678 | 0.701 | 0.715 | 0.682 | **0.835** |
| | **PLA** | 0.707 | 0.758 | 0.758 | 0.738 | **0.851** |

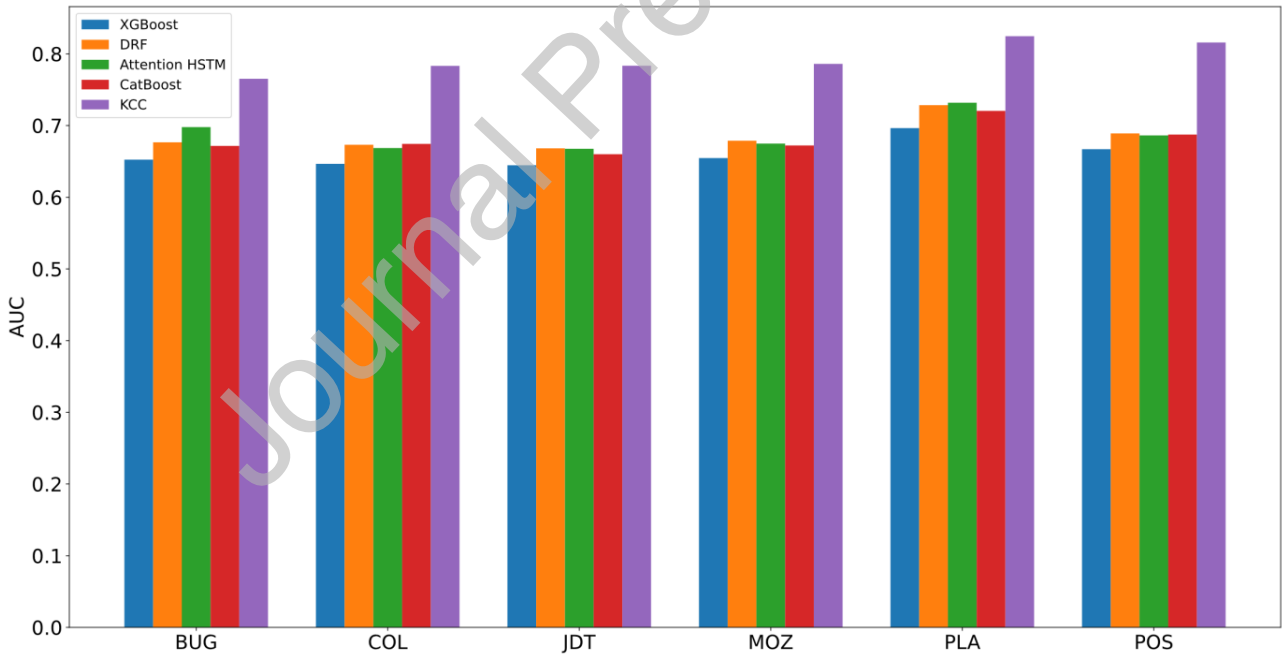


**Fig.5.** Comparison of the average AUC of the target projects between the KCC and baseline methods

**RQ5: Do the results obtained from our KCC method differ significantly using those of the baseline method based on statistical tests?**

The study used the Scott–Knott test to compare the KCC method with the baseline method and assess the presence of a statistically significant difference between them. The mean values of the G-mean, MCC, F1-score, and AUC metrics for the target project were compared. The comparison objects were ranked and grouped using the Scott–Knott test. Box plots were generated based on the values of the G-mean, MCC, F1-score, and AUC metrics for each target project, as shown in Figs. 6, 7, 8, and 9,

respectively.

The plots were categorized into three groups: red, blue, green, purple, and orange for the KCC, CatBoost, Attention HSTM, XGBoost, and DRF methods, respectively. Based on the statistical analysis shown in Figs. 6, 7, 8 and 9, our KCC method outperforms the baseline methods in terms of both the G-mean, MCC, F1-score, and AUC metrics for any given project. This difference is statistically significant, providing evidence of the superior performance of our KCC method.

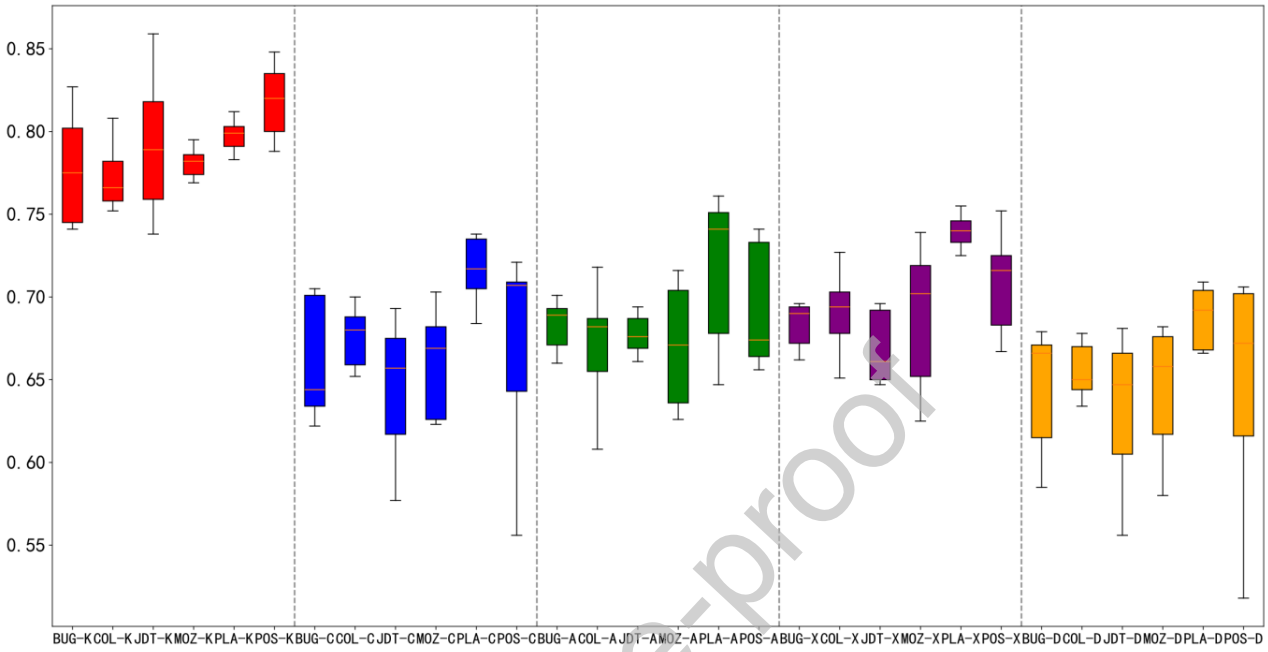


**Fig. 6.** Comparison of G-means for target projects between the KCC and baseline methods using the Scott–Knott test

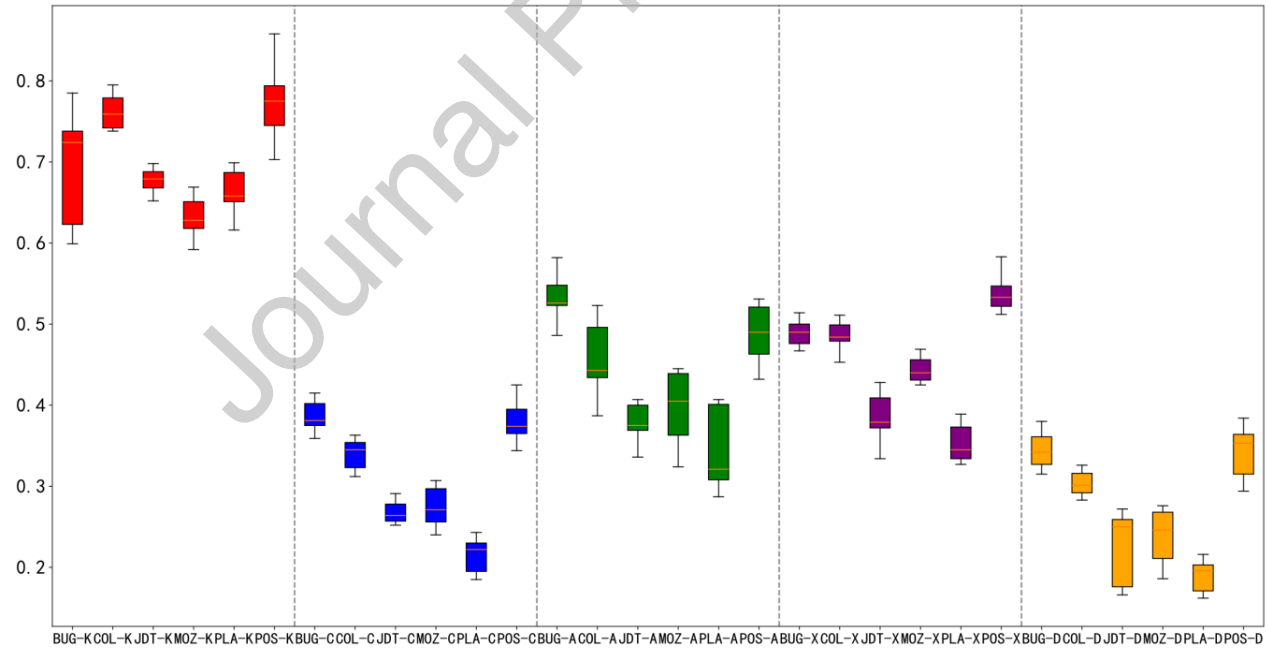


**Fig. 7.** Comparison of MCC for target projects between KCC and baseline methods using the Scott–Knott test

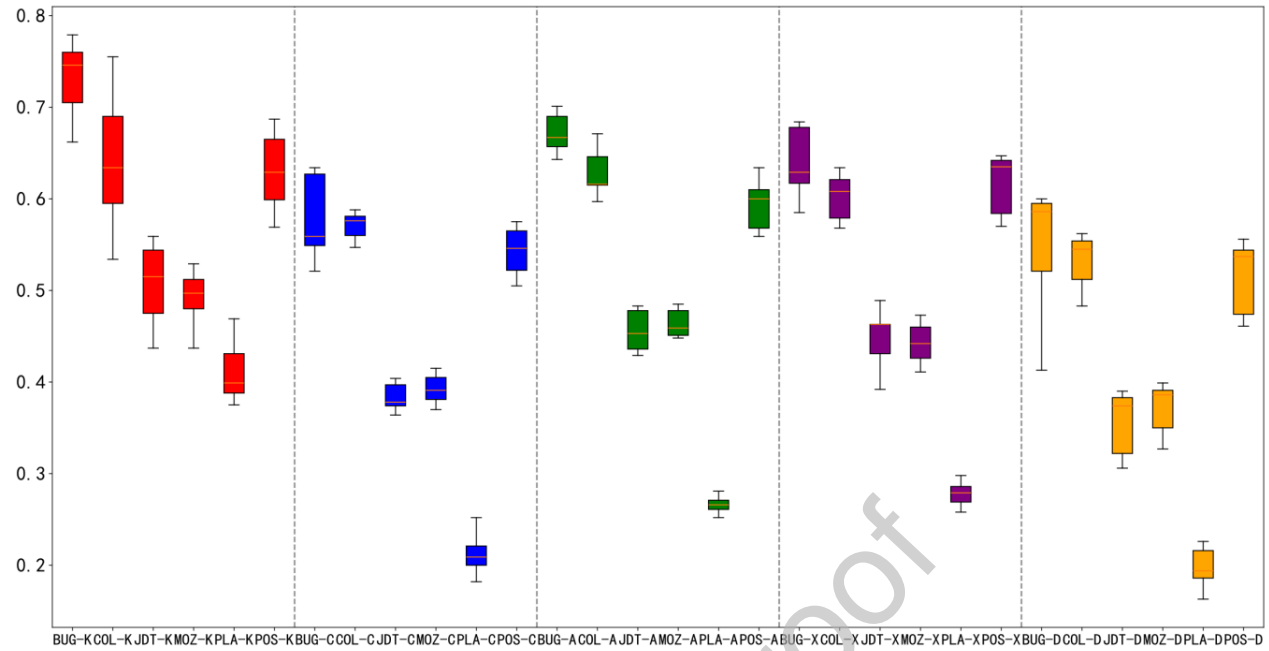


**Fig. 8.** Comparison of F1-score for target projects between KCC and baseline methods using the Scott–Knott test


**Fig. 9.** Comparison of AUC for target projects between KCC and baseline methods using the Scott–Knott test

## 6. Discussions

### *6.1. Evaluating the Effectiveness of KCC*

Tables 4, 5, 6, and 7 demonstrate that the KCC method has favorable transfer effects in terms of the G-mean, MCC, F1-score, and AUC metrics. Statistical testing showed that the KCC method was significantly superior to the baseline methods. To further support the effectiveness of the KCC method, we compared its prediction results with those of the within-project (WP) SDP. The CatBoost algorithm was used to construct the JIT-SDP model. The change dataset was randomly partitioned into training and testing sets with 70% and 30% of the data, respectively. The training set was balanced based on the data processing approach used in this study. The test set was left unbalanced. The JIT-SDP model was trained using the designated training set and was used to make predictions on the test set. To ensure the reliability of the findings, we iterated the procedure 10 times and used the average of these iterations as the

final prediction result. We compared the average G-mean, MCC, F1-score, and AUC metrics of the target projects generated by the KCC method with the values presented in Table 9. The comparison results are presented in Table 9, with the best values in each row highlighted in bold.

Table 9 shows that the G-mean of the KCC method is similar to that of the WP method. However, the average G-mean of the KCC method outperforms that of the WP method. Additionally, the MCC, F1-score, and AUC of the KCC method exceed that of the WP method. To conduct a more comprehensive comparison of the performance of the KCC and WP methods using these four metrics, a bubble chart is created. The chart depicts six projects on the x-axis and G-mean on the y-axis, with the size of the bubbles proportional to the MCC, F1-score, or AUC, as shown in Fig. 10. Based on the data shown in Fig. 10, it is clear that the KCC and WP methods perform similarly in terms of G-mean in the COL and MOZ projects. However, the KCC method significantly outperforms the WP method for the other four projects. This provides further evidence of the effectiveness of the KCC method.

### *6.2. Evaluating the Effectiveness of KVM*

To demonstrate the effectiveness of the KVM method, a comparison was conducted using the classical weighted adaptive method known as KMM [29]. The KMM method is widely used in various domains, such as computer vision [43], text classification [44], and causal inference [45], and it has demonstrated strong performance. This study compared the effectiveness of the KVM and KMM methods using the CatBoost algorithm and its default loss function to construct a JIT-SDP model. The KVM and KMM algorithms were integrated into the model, which used G-mean, MCC, F1-score, and AUC as comparison metrics for the target projects. To ensure reliable results, the process was iterated 10 times, and the average of these iterations was used as the final prediction. Box plots were created for each target project using the five predicted values for each metric, as shown in Figs. 11, 12, 13, and 14.

Based on the findings presented in Figs. 11, 12, 13, and 14, it is clear that the KVM method outperforms the KMM method across all metrics evaluated in the target project. Comparing Fig. 11 with Fig. 6, it is evident that both the KVM and KMM methods outperform the CatBoost method in terms of the G-mean metric. Comparing Fig. 12 with Fig. 7, it is clear that both the KVM and KMM methods outperform the CatBoost method in terms of the MCC metric. Comparing Fig. 13 with Fig. 8, it is evident that both the KVM and KMM methods outperform the CatBoost method in terms of the F1-score metric. Comparing Fig. 14 with Fig. 9, it is clear that both the KVM and KMM methods outperform the CatBoost method in terms of the AUC metric. This demonstrates that the KVM and KMM methods are effective in improving the performance of the G-mean, MCC, F1-score, and AUC for JIT-SDP. However, there is still a significant difference between the KVM and KCC methods.

In summary, the KVM method is shown to improve the performance of JIT-SDP. It is important to note that assuming identical conditional probability distributions for the two projects is invalid when adapting the marginal probability distributions between the source and target projects. Therefore, addressing only the marginal probability distribution problem using the KVM method to adapt the marginal probability distributions between the source and target projects may lead to the emergence of new conditional probability distribution problems. While improving performance, this is not entirely optimal.

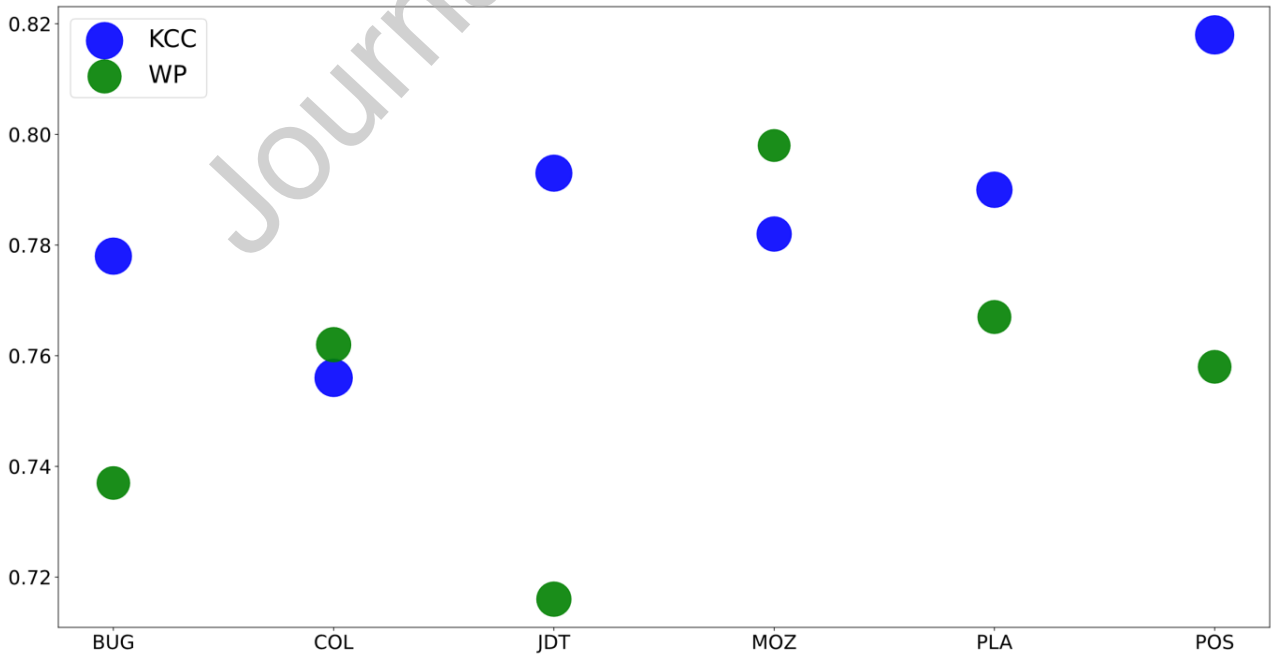


**Fig. 10.** Comparison of G-mean for target projects between KCC and Within-Project

**Table 9** KCC vs. Within-Project

| Target | KCC | | | | WP | | | |
|---|---|---|---|---|---|---|---|---|
| | G-Mean | MCC | F1-score | AUC | G-Mean | MCC | F1-score | AUC |
| **BUG** | **0.778** | **0.694** | **0.73** | **0.765** | 0.737 | 0.463 | 0.669 | 0.731 |
| **COL** | 0.756 | **0.738** | **0.642** | **0.783** | **0.762** | 0.422 | 0.626 | 0.694 |
| **JDT** | **0.793** | **0.688** | **0.506** | **0.783** | 0.716 | 0.322 | 0.426 | 0.71 |
| **MOZ** | 0.782 | **0.623** | **0.496** | **0.786** | **0.798** | 0.337 | 0.433 | 0.731 |
| **PLA** | **0.791** | **0.66** | **0.411** | **0.825** | 0.767 | 0.275 | 0.262 | 0.768 |
| **POS** | **0.818** | **0.775** | **0.63** | **0.816** | 0.758 | 0.466 | 0.614 | 0.753 |

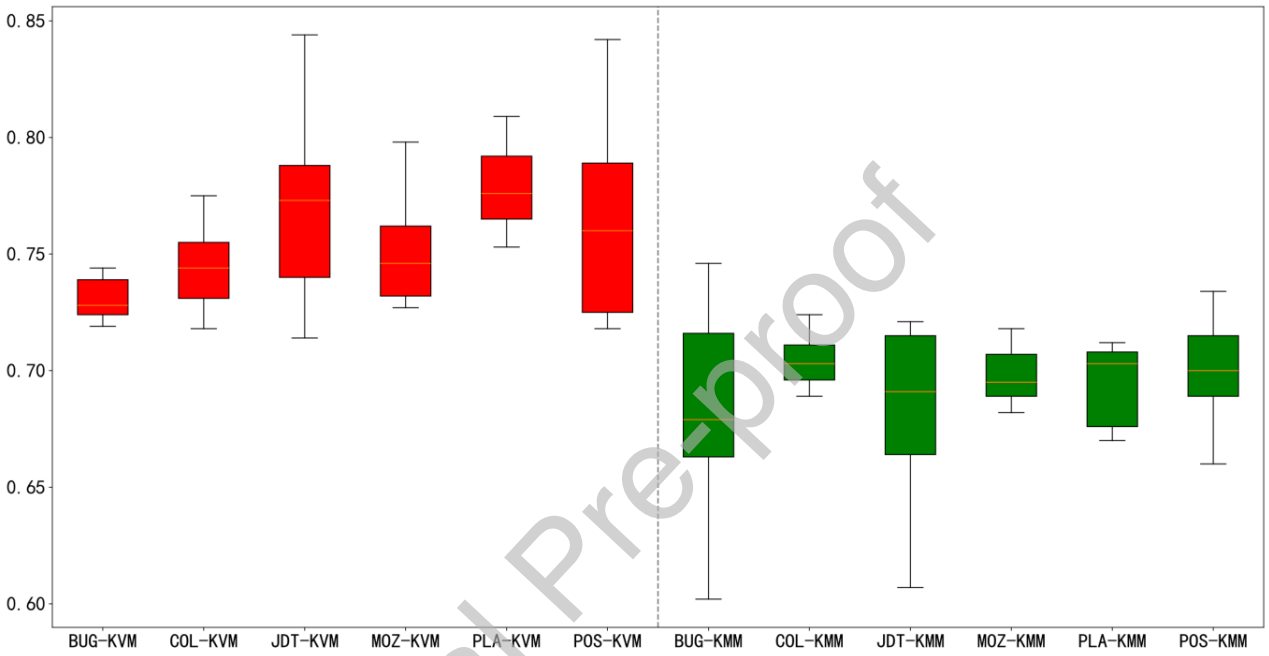


**Fig. 11.** Comparison of G-means for target projects between KVM and KMM

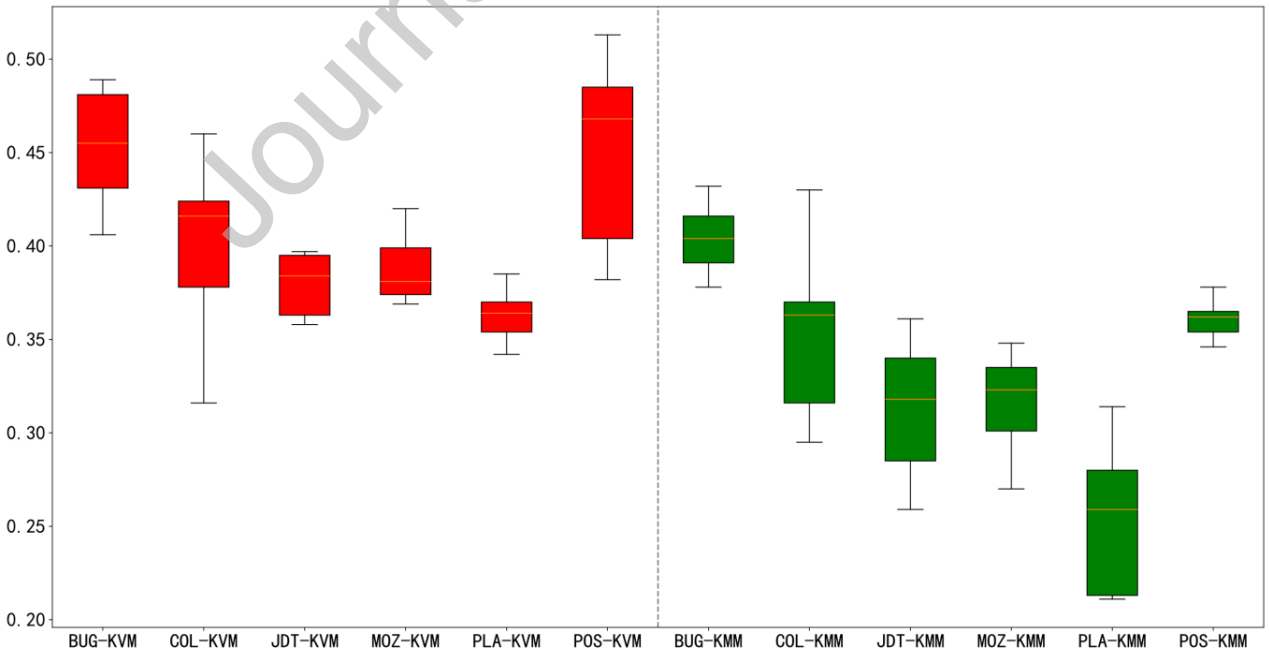


**Fig. 12.** Comparison of MCC for target projects between KVM and KMM

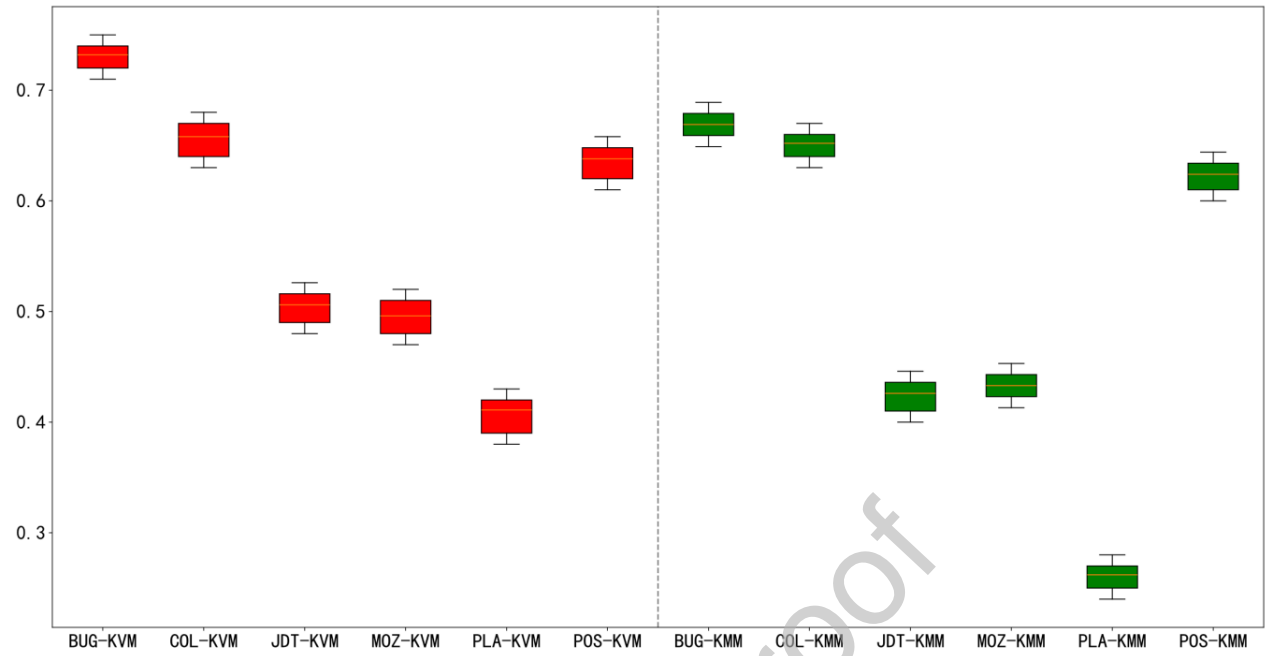


**Fig. 13.** Comparison of F1-score for target projects between KVM and KMM

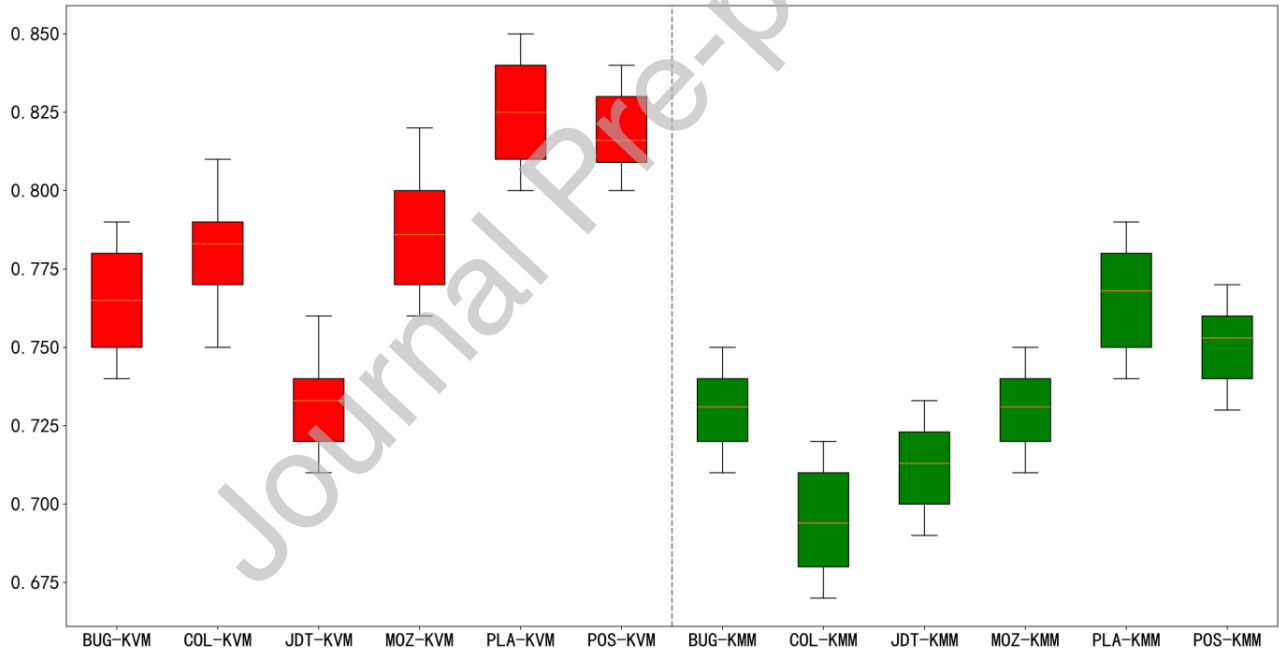


**Fig. 14.** Comparison of AUC for target projects between KVM and KMM

*6.3. Evaluating the Effectiveness of the Improved CORAL Loss Function*

The improved CORAL method is a simple implementation that does not require the specification of hyperparameters. It has shown significant advancements in domains such as image classification and object detection, as demonstrated by its positive outcomes [46]. In this study, the improved CORAL method was used to develop the loss function of the JIT-SDP model. To evaluate its effectiveness, the CatBoost algorithm was used to construct the JIT-SDP model. A comparison was conducted between the default loss function of the CatBoost algorithm and the loss function developed using the improved CORAL method. The JIT-SDP model was trained using the source project in both cases, and predictions were made for the target projects. The comparison metrics used were the G-mean and MCC of the target projects. To ensure the reliability of the results, the procedure was repeated 10 times, and the average of these 10 iterations was used as the final predictive result. Box plots were created for each target project, as shown in Figs. 15 and 16.

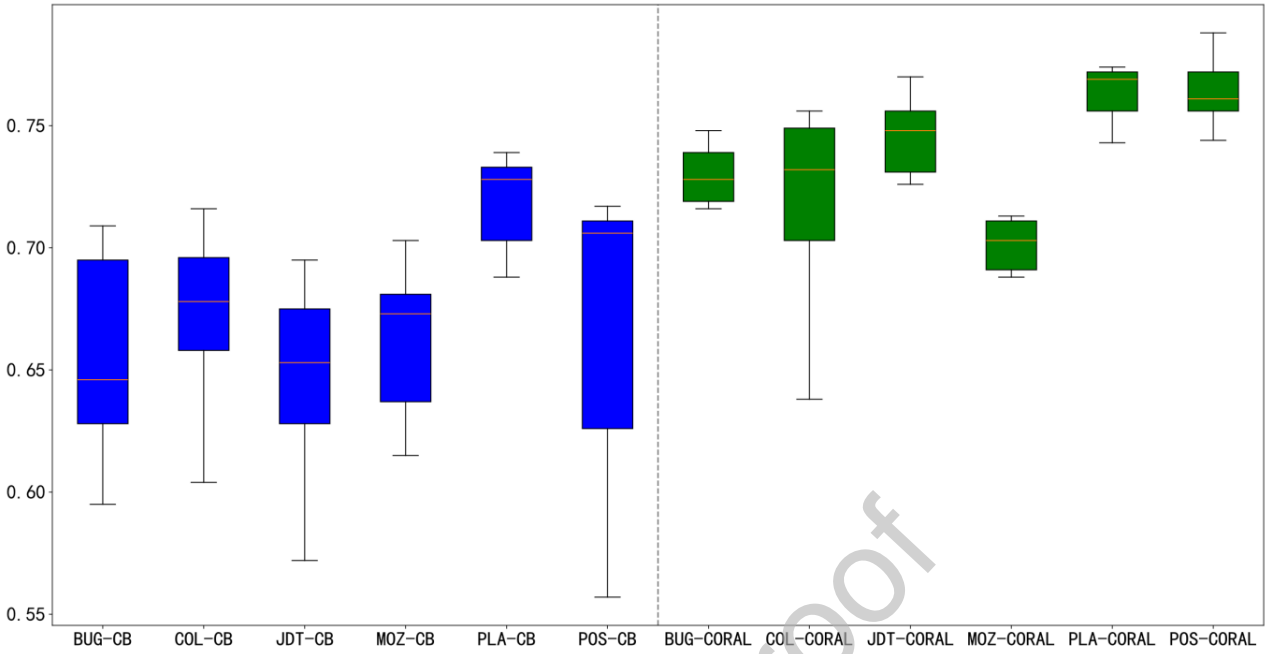


**Fig. 15.** Comparison of G-means for target projects between improved CORAL and CatBoost

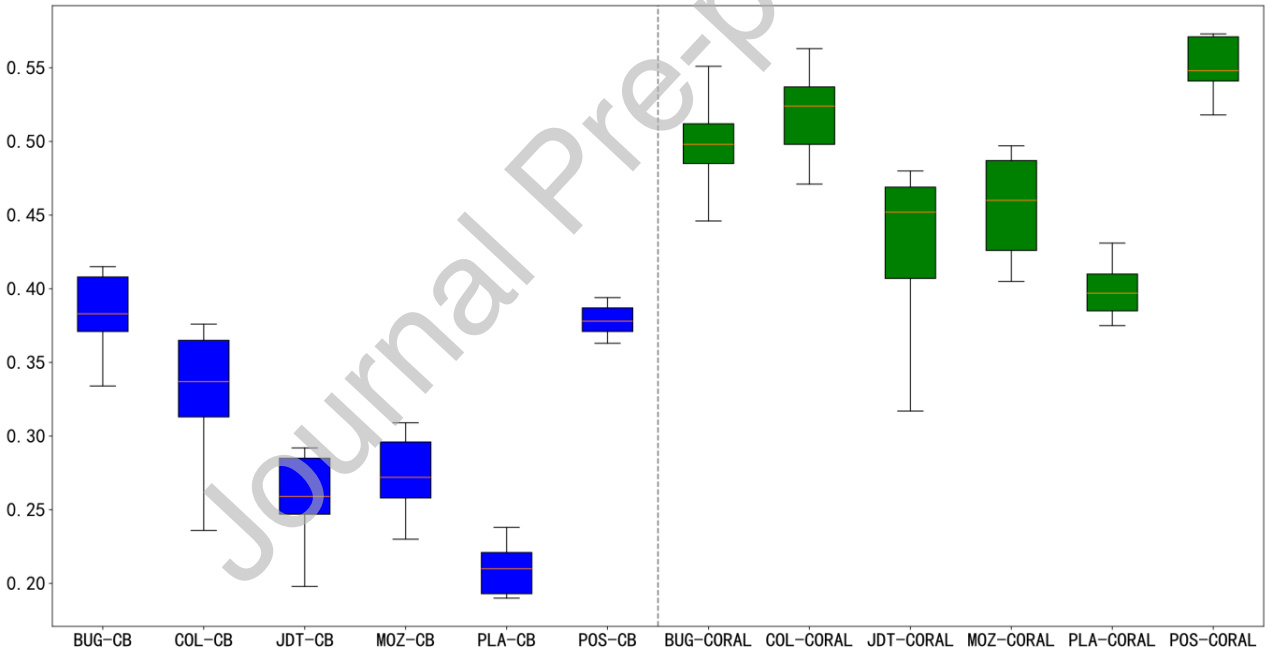


**Fig. 16.** Comparison of MCC for target projects between improved CORAL and CatBoost

Figs. 15 and 16 show that the improved CORAL method outperforms the CatBoost method in both the G-mean and MCC metrics. The use of the improved CORAL method as the loss function for the JIT-SDP model improved the G-mean and MCC performance. However, there is still a significant difference when compared with the KCC method. When considering the adjustment of the conditional probability distribution between the source and target projects, it is crucial to recognize that assuming identical marginal probability distributions for both projects is inappropriate. Therefore, when using the improved CORAL method to address the problem of adapting the conditional probability distribution between the source and target projects, it is important to consider the issues associated with the marginal probability distribution. Focusing solely on the conditional probability distribution may not be sufficient and could lead to suboptimal performance. However, some performance improvements can be achieved by addressing the marginal probability distribution.

In summary, the results suggest that adjusting the marginal or conditional probability distributions of the source and target projects can improve the performance of CP-JIT-SDP although the effect is suboptimal. This study used the KCC method to

achieve more optimal results for several main reasons. 1) The KVM method is incorporated to address the adaptation issue of the marginal probability distribution between the source and target projects. 2) The improved CORAL method is used to develop the model loss function to solve the adaptation problem of the conditional probability distribution between the source and target projects. 3) Both methods are dynamically integrated using the CatBoost algorithm, which optimizes the joint probability distribution of the source project to align more closely with that of the target project through multiple iterations. CatBoost's automatic feature handling and robustness to noise, outliers, and missing values, along with its other capabilities, enable the synergistic effects of KVM, improved CORAL, and CatBoost.

*6.4 Analysis of the Computational Complexity of KCC and the Baseline Methods*

This subsection presents an analysis of the computational complexity (CC) of the proposed method, KCC, and the baseline methods XGBoost, DRF, CatBoost, and Attention HSTM. The CC of these methods and the relevant factors affecting CC are presented in Table 10.

The results of Table 10 indicate that the CC of the KCC method is related to that of the KVM and improved CORAL methods, similar to the Attention HSTM method, but higher than that of the XGBoost, DRF, and CatBoost methods. Given that software change datasets are typically not extensive in scale, and servers possess sufficient computing power support, the actual computation time is not significantly different from that of the other methods. Consequently, it can be concluded that although the CC of the KCC method is high, it does not affect its practical application.

**Table 10** Computational Complexity of KCC vs. Baseline Methods

| Method | Relevant Factors | Computational Complexity |
|---|---|---|
| KVM | number of samples (n)<br>number of iterations (T) | $O(n^2 * T)$ |
| Improved CORAL | feature dimension (d)<br>number of source samples ($N_s$)<br>number of target samples ($N_t$) | $O(d*(N_s^2 + N_t^2) + d^3)$ |
| KCC | CC of KVM<br>CC of improved CORAL<br>number of iterations (T) | $(CC(KVM) + CC(improved\ CORAL)) * T$ |
| XGBoost | number of samples (n)<br>number of features (m)<br>depth of the tree (d) | $O(n*m*d)$ |
| DRF | number of samples (m)<br>feature dimension (n)<br>maximum depth of tree (d)<br>number of trees (T) | $O(m*n*d*T)$ |
| CatBoost | number of trees (T)<br>feature dimension (d)<br>number of samples (n) | $O(n*d*T)$ |
| Attention HSTM | number of samples (n)<br>feature dimension (d) | $O(n^2 * m)$ |

## 7. THREATS TO VALIDITY

*7.1. Internal Validity*

Internal validity is primarily associated with two factors.

1) Accuracy and reliability of the experimental implementation: To ensure the accuracy, reliability, and consistency of the experimental code, we implemented various measures such as standardized coding practices, team-based pair programming techniques, and thorough code reviews. Additionally, to improve the reliability of the results, we calculated the average values from 10 repeated runs.

2) Parameter configurations used in CatBoost: A stochastic search methodology was used to optimize the hyperparameters of the CatBoost algorithm.

The study took sufficient precautions to maintain internal validity, and no threats to it were identified.

*7.2. External Validity*

External validity in this study is mainly influenced by two factors.

1) The construction of the change dataset involves the following two crucial steps: measuring change metrics and identifying defect-causing changes. Measuring change metrics involves using counting, statistical, and other methodologies, while

identifying changes that cause defects requires the use of the SZZ algorithm or its variants. Both factors introduce potential errors that could compromise external validity.

2) The experimental implementation of this study was conducted using Python technology, which is known for its flexibility and openness. Although our approach has the potential to be applied to other projects, it is essential to verify whether the predicted results align with those obtained in our study.

In summary, external validity carries inherent risks that require scrutiny and validation on a project-specific basis.

### *7.3. Construct Validity*

The construct validity of the performance evaluation metrics in the JIT-SDP domain is closely connected to their appropriateness for measuring predictive performance. Commonly used metrics in this field include accuracy, precision, recall, F1-score, G-mean, AUC, and MCC. To evaluate the capability of the SDP model to accurately identify both defective and nondefective changes, and to measure the overall quality of the prediction results, we used G-mean, MCC, F1-score, and AUC as performance evaluation metrics. Our study carefully considers the construct validity by selecting appropriate performance evaluation metrics.

## 8. Conclusion and Future Work

This study presents a new method called KCC, which uses transfer learning with DDA to solve the issue of CP-JIT-SDP. This method prpposes the KVM approach to address the issue of differing marginal probability distributions between the source and target projects. This process involves adjusting the weight of the source project and calculating the variance in the RKHS between the source and target projects to harmonize their variances. The JIT-SDP models are constructed using the CatBoost algorithm, while the loss function of the model is formulated using the improved CORAL method. The loss function measures the difference between the covariance matrices of the source and target project datasets using the square of the Frobenius norm. The combined use of CatBoost and improved CORAL is utilized to address the issue of varying conditional probability distributions between the source and target projects. By dynamically integrating the source and target projects through several iterations, the joint probability distribution of the source project is optimized to align closely with that of the target project. Additionally, CatBoost's automatic feature handling and robustness to noise, outliers, and missing values, along with its other capabilities, enable the synergistic effects of KVM, improved CORAL, and CatBoost.

In this study, we conducted experiments on six well-known open-source projects to validate the effectiveness of the KCC method. In contrast to the KMM method [29], we confirmed the effectiveness of the KVM method in improving the performance of CP-JIT-SDP. Compared with the CatBoost method, our results confirm that using the improved CORAL method as a loss function for the JIT-SDP model is similarly effective in improving the performance of CP-JIT-SDP. However, using these two methods in isolation does not comprehensively improve the performance of CP-JIT-SDP. The KCC method significantly improves the CP-JIT-SDP by dynamically integrating the KVM method, improved CORAL method, and CatBoost algorithm. The average rate of improvement in G-mean is 18%, that of MCC is 105.4%, that of F1 score is 25.6%, and that of AUC is 16.9%. The KCC method exhibits exceptional stability.

In future research, we will extend the KCC method to a broader range of open-source and commercial projects to verify its effectiveness. Additionally, we will explore new optimization methods to enhance the performance of the KCC method. Moreover, we will continue investigating transfer learning with DDA and its applications across various domains.

## Acknowledgment

This work was supported by several funding sources, including Guangdong Province Key Area Research and Development Plan Project (2023B0202120001), the National Social Science Foundation Major Project of China (23&ZD127), Guangdong Basic and AppliedBasic Research Foundation (2024A1515010111,2023A1515011520),Tertiary Education Scientiffc Research Project of Guangzhou Municipal Education Bureau (202235324), and Guangdong Provincial Higher Education Teaching Reform Project (Yue Jiao Gaohan [2021] No.29).

## References

[1] Kumar, S., & Rathore, S. (2018). Software Fault Prediction: A Road Map.

[2] Giger, E., D'Ambros, M., Pinzger, M., & Gall, H. C. (2012). Method-level bug prediction. Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Presented at the ESEM '12: 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Lund Sweden.

[3] Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting Defects for Eclipse. Predictive Models in Software Engineering, Predictive Models in Software Engineering.

[4] Nagappan, N., & Ball, T. (2005). Use of relative code churn measures to predict system defect density. Proceedings of the 27th International Conference on Software Engineering - ICSE '05. Presented at the 27th International Conference, St. Louis, MO, USA.
[5] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2013). A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering, 757–773.
[6] Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, 1345–1359.
[7] Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. Information and Software Technology, 248–256.
[8] Nam, J., Pan, S., & Kim, S.-H. (2013). Transfer defect learning. International Conference on Software Engineering, International Conference on Software Engineering.
[9] Liu, C., Yang, D., Xia, X., Yan, M., & Zhang, X. (2019). A two-phase transfer learning model for cross-project defect prediction. Information and Software Technology, 125–136.
[10] Tang, S., Huang, S., Zheng, C., Liu, E., Zong, C., & Ding, Y. (2022). A novel cross-project software defect prediction algorithm based on transfer learning. Tsinghua Science and Technology, 41–57.
[11] Yang, Q., Zhang, Y., Dai, W., & Pan, S. J. (2020). Transfer Learning.
[12] Wang, J., Chen, Y., Feng, W., Yu, H., Huang, M., & Yang, Q. (2020). Transfer Learning with Dynamic Distribution Adaptation. ACM Transactions on Intelligent Systems and Technology, 1–25.
[13] Sun, B., Feng, J., & Saenko, K. (2022). Return of Frustratingly Easy Domain Adaptation. Proceedings of the AAAI Conference on Artificial Intelligence.
[14] Mockus, A., & Weiss, D. M. (2002). Predicting risk of software changes. Bell Labs Technical Journal, 169–180.
[15] McIntosh, S., & Kamei, Y. (2018). Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. IEEE Transactions on Software Engineering, 412–428.
[16] Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. Information and Software Technology, 204–216.
[17] Zhao, K., Xu, Z., Zhang, T. Z., Tang, Y., & Yan, M. (2021). Simplified Deep Forest Model-Based Just-in-Time Defect Prediction for Android Mobile Apps. IEEE Transactions on Reliability, 848–859.
[18] Ardimento, P., Aversano, L., Bernardi, M. L., Cimitile, M., & Iammarino, M. (2022). Just-in-time software defect prediction using deep temporal convolutional networks. Neural Computing and Applications, 3981–4001.
[19] Cabral, G. G., & Minku, L. L. (2023). Towards Reliable Online Just-in-Time Software Defect Prediction. IEEE Transactions on Software Engineering, 49, 1342-1358.
[20] He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering, 19(2), 167–199.
[21] Pan, S. J., Tsang, I. W., Kwok, J. T., & Yang, Q. (2011). Domain adaptation via transfer component analysis. IEEE Transactions on Neural Networks, 199–210.
[22] Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., & Hassan, A. E. (2016). Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, 2072–2106.
[23] Xia, X., Lo, D., Pan, S. J., Nagappan, N., & Wang, X. (2016). HYDRA: Massively Compositional Model for Cross-Project Defect Prediction. IEEE Transactions on Software Engineering, 977–998.
[24] Herbold, S., Trautsch, A., & Grabowski, J. (2018). A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches. IEEE Transactions on Software Engineering, 811–833.
[25] Zhu, K., Zhang, N., Ying, S., & Zhu, D. (2020). Within- project and cross- project just- in- time defect prediction based on denoising autoencoder and convolutional neural network. IET Software, 185–195.
[26] Lin, D., Tantithamthavorn, C., & Hassan, A. E. (2022). The Impact of Data Merging on the Interpretation of Cross-Project Just-In-Time Defect Models. IEEE Transactions on Software Engineering, 2969–2986.
[27] Tabassum, S., Minku, L. L., & Feng, D. (2023). Cross-Project Online Just-In-Time Software Defect Prediction. IEEE Transactions on Software Engineering, 268–287.
[28] Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, 1345–1359.
[29] Huang, J., Gretton, A., Borgwardt, K., Schölkopf, B., & Smola, A. (2007). Correcting Sample Selection Bias by Unlabeled Data. In Advances in Neural Information Processing Systems 19.
[30] Wang, J., Chen, Y., Hu, L., Peng, X., & Yu, P. S. (2018). Stratified Transfer Learning for Cross-domain Activity Recognition. 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom). Presented at the 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom), Athens.
[31] Pandey, S. K., & Tripathi, A. K. (2023). Cross-Project setting using Deep learning Architectures in Just-In-Time Software Fault Prediction: An Investigation. 2023 IEEE/ACM International Conference on Automation of Software Test (AST). Presented at the 2023 IEEE/ACM International Conference on Automation of Software Test (AST), Melbourne, Australia.

[32] Tang, S., Huang, S., Zheng, C., Liu, E., Zong, C., & Ding, Y. (2022). A novel cross-project software defect prediction algorithm based on transfer learning. Tsinghua Science and Technology, 41–57.
[33] Jiang, J., & Zhai, C. (2007). Instance Weighting for Domain Adaptation in NLP. Meeting of the Association for Computational Linguistics, Meeting of the Association for Computational Linguistics.
[34] Gretton, A., Borgwardt, K. M., Rasch, M., Schölkopf, B., & Smola, A. J. (2007). A Kernel Method for the Two-Sample-Problem. In Advances in Neural Information Processing Systems 19 (pp. 513–520).
[35] da Costa, D. A., McIntosh, S., Shang, W., Kulesza, U., Coelho, R., & Hassan, A. E. (2017). A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes. IEEE Transactions on Software Engineering, 641–657.
[36] Fan, Y., Xia, X., da Costa, D. A., Lo, D., Hassan, A. E., & Li, S. (2021). The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction. IEEE Transactions on Software Engineering, 1559–1586.
[37] Goyal, J., & Ranjan Sinha, R. (2022). Software Defect-Based Prediction Using Logistic Regression: Review and Challenges. In Second International Conference on Sustainable Technologies for Computational Intelligence, Advances in Intelligent Systems and Computing (pp. 233–248).
[38] Qiu, S., Lu, L., & Jiang, S. (2019). Joint distribution matching model for distribution–adaptation- based cross- project defect prediction. IET Software, 13(5), 393–402.
[39] Azzeh, M., Elsheikh, Y., Nassif, A. B., & Angelis, L. (2023). Examining the performance of kernel methods for software defect prediction based on support vector machine. Science of Computer Programming, 226, 102916.
[40] Tabassum, S., Minku, L. L., Feng, D., Cabral, G. G., & Song, L. (2020). An investigation of cross-project learning in online just-in-time software defect prediction. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Presented at the ICSE '20: 42nd International Conference on Software Engineering, Seoul South Korea.
[41] Yao, J., & Shepperd, M. (2021). The impact of using biased performance metrics on software defect prediction research. Information and Software Technology, 139, 106664.
[42] Jelihovschi, E., Faria, J. C., & Allaman, I. B. (2015). ScottKnott: A Package for Performing the Scott-Knott Clustering Algorithm in R. TEMA (São Carlos), 15(1), 003.
[43] Wang, Y., Zhao, D., Li, Y., Chen, K., & Xue, H. (2019). The Most Related Knowledge First: A Progressive Domain Adaptation Method. In Lecture Notes in Computer Science, Trends and Applications in Knowledge Discovery and Data Mining (pp. 90–102).
[44] Wang, Z., Bi, W., Wang, Y., & Liu, X. (2019). Better Fine-Tuning via Instance Weighting for Text Classification. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 7241–7248.
[45] Moraffah, R., Shu, K., Raglin, A., & Liu, H. (2019). Deep causal representation learning for unsupervised domain adaptation. Cornell University - arXiv.
[46] Li, Y., Wang, N., Shi, J., Liu, J., & Hou, X. (2016). Revisiting Batch Normalization for Practical Domain Adaptation. arXiv: Computer Vision and Pattern Recognition.
[47] Huang, Q., Xia, X., & Lo, D. (2019). Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. Empirical Software Engineering, 2823–2862.
[48] Chen, X., Zhao, Y., Wang, Q., & Yuan, Z. (2018). MULTI: Multi-objective effort-aware just-in-time software defect prediction. Information and Software Technology, 1–13.
[49] Zhang, T., Yu, Y., Mao, X., Lu, Y., Li, Z., & Wang, H. (2022). FENSE: A feature-based ensemble modeling approach to cross-project just-in-time defect prediction. Empirical Software Engineering, 27(7).
[50] Y. Wang, Y. Li, Y. Ren, & J. Yu. (2023). Enhancing Cross-Project Just-In-Time Defect Prediction with Active Deep Learning. 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C), Chiang Mai, Thailand, 2023, pp. 93-102.
[51] Zheng, S., Gai, J., Yu, H., Zou, H., & Gao, S. (2021). Training data selection for imbalanced cross-project defect prediction. Computers and Electrical Engineering, 94, 107370.
[52] Jiang, Y., Cukic, B., & Ma, Y. (2008). Techniques for evaluating fault prediction models. Empirical Software Engineering, 13(5), 561–595.
[53] Craven, M., & Bockhorst, J. (2004). Markov Networks for Detecting Overlapping Elements in Sequence Data.
[54] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research.
[55] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. Le Centre Pour La Communication Scientifique Directe - HAL - Inria.
[56] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Presented at the KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco California USA.
[57] Han, S., Kim, H., & Lee, Y.-S. (2020). Double random forest. Machine Learning, 109(8), 1569–1586.
[58] Ganaie, M. A., Tanveer, M., Suganthan, P. N., & Snasel, V. (2022). Oblique and rotation double random forest. Neural Networks, 153, 496–517.

[59] Liu, J., Wang, G., Duan, L.-Y., Abdiyeva, K., & Kot, A. C. (2018). Skeleton-Based Human Action Recognition With Global Context-Aware Attention LSTM Networks. IEEE Transactions on Image Processing, 1586–1599.
[60] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. ,arXiv: Learning.
[61] Ali, A. H., Yaseen, M. G., Aljanabi, M.,& Abed, S. A. (2023). Transfer Learning: A New Promising Techniques. Mesopotamian Journal of Big Data, 29–30.

Hongming Dai, Ph.D. and associate professor at the School of Software Engineering, South China University of Technology. He is also a nationally certified systems analyst, system architecture designer, and information systems project manager. He is a member of the CCF. He earned a Bachelor's degree in Computer Science and Technology from Wuhan University of Technology in 2000 and went on to complete his Master's degree in Software Engineering from Sun Yat-sen University in 2008. His research achievements include 18 academic papers published in renowned domestic and international journals such as Knowledge-Based Systems, Computer Science, and Computer Education. In addition to his publications, he has authored an academic book and holds seven authorized national patents, including two invention patents, two utility model patents, and three design patents. He has also applied for eight software copyrights and obtained four software application certificates. He has led seven national and provincial-level research projects and participated as a key contributor in 13 additional projects at the same level. He has also mentored students who have won more than 100 competition awards. He has received four teaching achievement awards and two teaching achievement awards as the first author. His primary research interests include machine learning, big data analysis, and software engineering.

Jianqing Xi, Ph.D., professor, and doctoral supervisor at the School of Software Engineering, South China University of Technology. He has undertaken two projects funded by the Natural Science Foundation of Guangdong Province, seventeen projects funded by the Science and Technology Department of Guangdong Province, two projects funded by the Science and Technology Department of Guangzhou City, and twelve projects commissioned by enterprises and institutions. He has published more than 80 papers, including over 20 papers indexed in SCI and EI databases. He has obtained four invention patents and fifteen software copyrights. He has been awarded one second prize and two third prizes of the Science and Technology Award of Guangzhou City.

Hong-Liang Dai, Ph.D., professor of the Department of Statistics of Guangzhou University, a distinguished professor of Guangzhou Scholars, doctoral supervisor, post-doctoral cooperative supervisor, and head of Data Science and Big Data Technology major. In 2003, he graduated from Wuhan University with a master's degree in Applied Mathematics. In June 2013, he graduated from Sun Yat-sen University with a major in Applied Mathematics. In 2015, he was awarded the title of Professor of Artificial Intelligence. In recent years, he has been working in the fields of machine learning and big data analysis. He has published more than 40 papers in internationally renowned journals such as Knowledge-Based Systems, Applied Soft Computing, Computers & Industrial Engineering, and Neurocomputing. He has also published a monograph with a prestigious international publishing house. In addition, he has applied for three invention patents, two utility patents, and holds four software copyrights. Anonymous reviewers evaluated articles for several academic journals, including IEEE Transactions on Neural Networks and Learning Systems, IEEE Transactions on Fuzzy Systems, Pattern Recognition, and Information Sciences.

**Declaration of interests**

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

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: