Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Multi-task Defect Prediction

Chao Ni[1] | Xiang Chen[1,2] | Xin Xia[3] | Qing Gu[1] | Yingquan Zhao[2]

[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2]School of Computer Science and Technology, Nantong University, Nantong, China
[3]Faculty of Information Technology, Monash University, Melbourne, Australia

Correspondence
Qing Gu, State Key Laboratory for Novel Software Technology, Nanjing University.
Email: guq@nju.edu.cn
Xiang Chen, School of Computer Science and Technology, Nantong University.
Email: xchencs@ntu.edu.cn

Present Address
State Key Laboratory for Novel Software Technology, Nanjing University.

Summary

Within-project defect prediction assumes that we have sufficient labeled data from the same project, while cross-project defect prediction assumes that we have plenty of labeled data from source projects. However, in practice, we might only have limited labeled data from both the source and target projects in some scenarios. In this paper, we want to apply multi-task learning to investigate such a new scenario. To our best knowledge, this problem (i.e., both the source project and the target project have limited labeled data) has not been thoroughly investigated and we are the first to propose a novel multi-task defect prediction approach MASK. MASK consists of a differential evolution optimization phase and a multi-task learning phase. The former phase aims to find optimal weights for shared and non-shared information in related projects (i.e., the target project and its related source projects), while the latter phase builds prediction models for each project simultaneously. To verify the effectiveness of MASK, we perform experimental studies on 18 real-world software projects and compare our approach with four state-of-the-art baseline approaches: STL, SCL, Peters filter and Burak filter. Experimental results show that MASK can achieve F1 of 0.397 and AUC of 0.608 on average with a few labeled data (i.e., 10% of data). Across the 18 projects, MASK can outperform baseline methods significantly in terms of F1 and AUC. Therefore, by utilizing the relatedness among multiple projects, MASK can perform significantly better than the state-of-the-art methods. The results confirm that MASK is promising for software defect prediction when the source and target projects both have limited training data.

KEYWORDS:
Software Defect Prediction, Multi-Task Learning, Differential Evolution, Empirical Studies

1 | INTRODUCTION

Software defect prediction (SDP)[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15] is a hot research topic in current software engineering research domain and it can help to optimize test resource allocation by predicting defect-prone modules (the granularity of the module can be set as file, class, code change as needed) in advance[16]. A number of defect prediction approaches have been proposed,

and these approaches mainly apply machine learning techniques to build prediction models by mining data stored in software historical repositories[17,18,19,20,21]. These approaches typically use various features (i.e., metrics) to measure extracted modules from repositories and then apply machine learning algorithms to predict if a new module is defective or not. Most of the proposed approaches work on within-project defect prediction (WPDP) scenario, i.e., the prediction models are trained and then applied to modules from the same project. These WPDP approaches require sufficient labeled (historical) data from a project.

However, in practice, it is rare that sufficient labeled data is available for a new project. Thus, researchers focus on cross-project defect prediction (CPDP) scenario[22,23,24,11,25,26], which builds a model using labeled data from other projects (i.e., source projects) to predict defective modules in a particular project (i.e., target project). Considering the domain difference phenomenon between the source and target projects, a defect prediction model trained on some projects might not generalize well to other projects[27]. Thus, transfer learning approaches, which aim to alleviate the differences between different projects, have been widely used to address the CPDP problem[22,23,24,11,25,26,28].

In the WPDP scenario, we assume that we have sufficient labeled data from the same project. In the CPDP scenario, we assume that we have plenty of labeled data from other projects. However, in practice, we might only have limited labeled data from both the source and target projects. Considering the following usage scenario:

Usage Scenario. Bob joined a start-up company on blockchain, and the company has a number of new projects. Bob was asked to identify defective files in these projects. One challenge was that all of these projects have limited labeled data. Thus, Bob performed cross-project defect prediction by using source projects from PROMISE[†] repository. However, preliminary results showed that the performance is very low due to the fact that their projects worked on different domains when compared with the projects from PROMISE repository. After discussing with the technical director, they decided to only use the projects in their company to perform defect prediction. Then, the core problem they meet is that:

How to perform defect prediction when both the source and target projects have limited labeled data?

In the machine learning community, multi-task learning (MTL)[29,30,31,32,33,34,35] is a kind of technique which can be used to solve this problem. MTL is one of the most active research topics in machine learning community. It aims to leverage useful information contained in multiple tasks to help improving the generalization performance of all the tasks. Among these learning tasks, a subset or all of them are assumed to be related to each other. Fortunately, there exits some relatedness between different projects. For example, projects from the same department in the organization may have the same code style and code specification.

[†]http://openscience.us/repo

In this paper, we propose a novel defect prediction approach MASK (Multi-tASK defect prediction) based on MTL. Our approach can effectively address the issue of training data insufficiency and make an improvement on performance. By using the information from related projects, each project equally has enough labeled modules. MASK builds models for each learned project by extracting common knowledge among all projects and personalized knowledge for each project. Besides, differential evolution optimization[36] is used by MASK to search for the optimal value of regularization parameters for the common knowledge and the personalized knowledge during the process of prediction model building.

To verify the effectiveness of our proposed MASK, we compare our approach with four state-of-the-art baseline approaches: STL (simple task learning), SCL (simple combined learning) proposed by Zimmermann et al.[27], Peters filter proposed by Peters et al.[37] and Burak filter proposed by Turhan et al.[16]. We use 18 widely used open-source projects, which contain a total of 9,393 modules, as our empirical subjects. Experimental results show that MASK can obtain the best performance on average. In particular, MASK achieves F1 of 0.397 and AUC of 0.608 on average when only 10% of data is labeled. Across the 18 projects, MASK improves the F1 of STL, SCL, Peters filter and Burak filter by 9.07%, 49.25%, 33.67% and 35.96% on average, and AUC of these baseline approaches by 23.42%, 36.85%, 32.20% and 16.52% at most, respectively. Moreover, results of statistical testing based on Wilcoxon signed-rank test with a Bonferroni correction show that the improvements of our proposed method are statistically significant and the effect sizes are non-negligible. We also investigate the influence of different percentages of labeled modules on the performance of MASK and we find that the more high-quality labeled modules, the higher performance of MASK.

Based on the above analysis, we find that treating multiple related projects simultaneously can effectively increase the size of training set for each project by considering the other projects and further improve the prediction performance. Therefore, MASK is a promising solution to solve the problem in our proposed new usage scenario that both the source and target projects have limited training data. It is not hard to find that the performance of MASK still has a certain distance from the satisfactory result. However, this is the first paper to address the issue of insufficient labeled modules by using multi-task learning and our study can encourage more researchers to propose more novel solutions based on MTL to further improve the performance.

The main contributions of the paper can be summarized as follows:

- To our best knowledge, we are the first to propose a new usage scenario for software defect prediction. In this scenario, both the source and target projects have limited labeled data. To solve this issue, we propose a novel approach MASK, which utilizes the advantages of differential evolution optimization and multi-task learning to build prediction models simultaneously for related projects.

- We conduct empirical studies based on real-world software projects to demonstrate the effectiveness of MASK. The final experimental results show that our approach can achieve a significant improvement over state-of-the-art baseline approaches.

The rest of this paper is organized as follows: Section 2 introduces the motivation and preliminary of multi-task learning. Section 3 describes our proposed MASK in detail. Section 4 introduces the empirical study, including experimental setup, the datasets, research questions, and results analysis. Section 5 discusses the limitation and computational cost of MASK. Section 6 briefly reviews related work, and finally Section 7 concludes the paper and discusses some future work.

2 | MOTIVATION AND BACKGROUND

In this section, we first show the motivation of using multi-task learning in our proposed new scenario, which the source and the target projects only have limited labeled data. Then we introduce the preliminary of multi-task learning to facilitate the subsequent introduction of our proposed approach.

2.1 | Motivation

In practice, both the source and target projects may have limited labeled data, which hinders the use of WPDP and CPDP. An example of this usage scenario can be found in Section 1. Therefore, a new approach to solve the problem in this usage scenario is urgently required. Multi-Task Learning is such an ideal candidate technique, which treats related projects simultaneously by extracting and utilizing appropriate shared information among them. On the contrary, previous studies [38,11,39,40,37,41] simply treated these projects independently and ignored the relatedness among projects, which can be called as Single-Task Learning (STL). Figure. 1 illustrates the difference between STL and MTL. For STL, each project is considered to be independent and learned independently. For MTL, multiple projects are considered together and learned simultaneously by utilizing the relatedness of these projects. Notice that by extracting and utilizing appropriate shared information among related projects, MTL outputs multiple models since each project also has its characteristics. Therefore, building multiple personalized models rather than one generalized model can make full use of characteristics and may have the potential to improve the performance.

MTL is inspired by human learning activities, in which, people often apply knowledge learned from previous tasks to learn a new task. For example, when a person learns to ride the bicycle and tricycle together, the experience in learning to ride the bicycle can be utilized in riding the tricycle and vice versa, since the bicycle and the tricycle are related to each other. Therefore, similar to human learning, it is useful for multiple learning tasks to be learned jointly since the knowledge contained in a task can be leveraged by other tasks. Taking software defect prediction as another example, when many new projects need defect

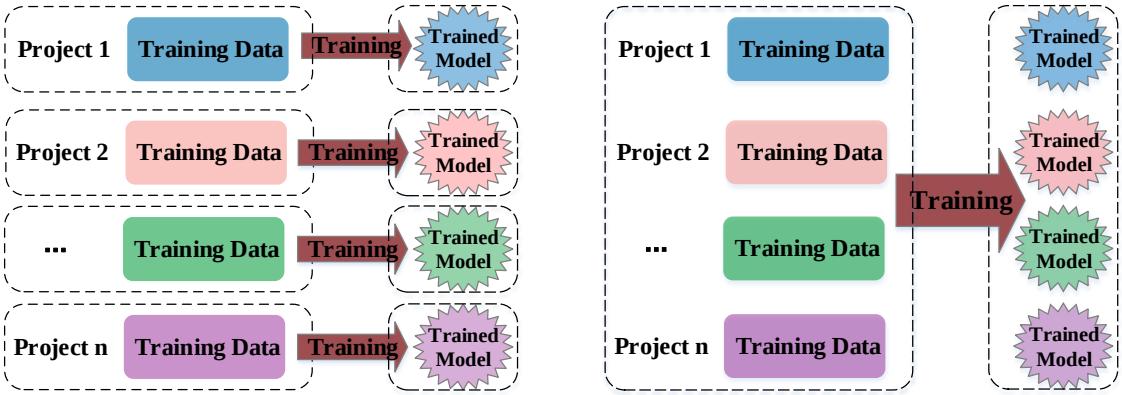

FIGURE 1 Comparison Between STL (Left) and MTL (Right).

prediction and these projects are related (e.g., these projects are developed by the same company for the same target), the rules learned in building a model on one project can be utilized to build a model on another project and vice versa.

The setting of MTL is similar to that of CPDP which is a specific case of transfer learning[42]. However, there are some differences between them. In MTL, there is no distinction among different tasks and the objective is to improve the performance of all related tasks. That is, all tasks in MTL are treated fairly. In CPDP, the source project and the target project are used to distinguish the roles of different projects, and the objective is to improve the performance of a target project with the help of the source project. That is, the target project plays a more important role than the source project. Hence, MTL treats all the projects equally but in CPDP the target project attracts more attention among all the projects.

Different from previous studies[38,11,37] which treat each project independently, in this paper, we treat all these projects fairly and simultaneously.

2.2 | Preliminary

In this subsection, we give some definitions of MTL to illustrate our proposed MASK more clearly. More details can be found in a recent survey for MTL[30].

Definition 1. (Multi-Task Learning) Suppose there are m tasks to be learned $\{T^i\}_{i=1}^m$ where all the tasks or at least a subset of them are related, multi-task learning aims to improve the learning of a model for T^i by using the knowledge contained in the m tasks.

In this paper, we focus on supervised learning based tasks since most studies for SDP fall into this setting. Notice that a project in SDP context is equally viewed as a task in the context of MTL. When different projects consider the same feature space, this setting is named as the homogeneous-feature MTL and otherwise it corresponds to heterogeneous-feature MTL. In this paper, the MTL setting is the homogeneous-feature MTL, which means all related tasks (i.e., projects) have the same feature space.

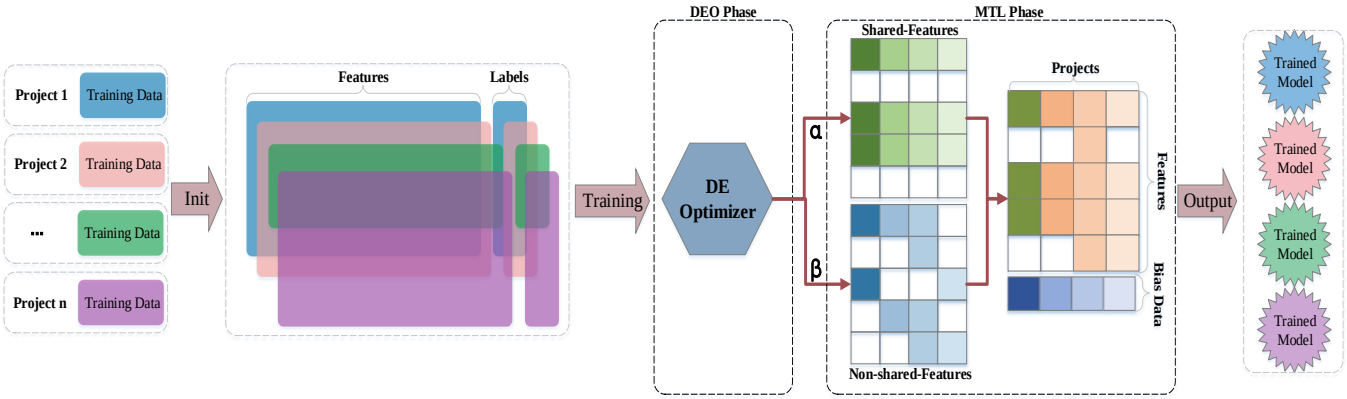

FIGURE 2 The Framework of MASK.

Definition 2. (Multiple Classification) The simple multiple classification model (i.e., a special scene of MTL) can be represented as the following: $Logistic\ Function\ (X^iW^i+Bias^i)$, $i = 1, \cdots, m$, which can build multiple classification models simultaneously.

In this definition, $X^i \in \mathbb{R}^{n^i \times F^i}$ represents the training module matrix of the i-th task, where n^i and F^i represent the number of modules and the feature space respectively. $Bias^i \in \mathbb{R}^{n^i}$ is a bias vector, and $Logistic\ Function\ (X^iW^i + Bias^i)$ represents the classification model. There are m tasks in total. For the convenience of description, we transform these quantities into matrices. That is, $W \in \mathbb{R}^{F^i \times m}$ represents the regression parameters and $Bias \in \mathbb{R}^{n^i \times m}$ represents the noise.

According to the above definitions, it is not hard to find that the parameter W is critically important since it contains the relatedness among different tasks. For convenience, we represent the parameters W as a matrix, where each column corresponds to a task, and each row to a feature. Meanwhile, not only do different tasks have a certain common knowledge (i.e., shared information), but also they have personalized knowledge (i.e., non-shared information). Therefore, both the shared information and non-shared information are included in the parameter W. It seems to be that any one structure might not fully capture all of that information, but decomposition of structure might[31]. Therefore the matrix W should be decomposed into two component matrices $Shared$ and $NonShared$, i.e., $W = Shared + NonShared$. In particular, some rows of W would have many non-zero entries which correspond to features shared by several projects. We use a row-sparse[31] matrix $Shared$ to represent such shared information, where each row is either all zero or mostly non-zero, and the number of non-zero rows is small. Some rows of W would be project-sensitive which correspond to those features related to some projects but not all. We use an elementwise spare matrix $NonShared$ to represent such non-shared information. It is obvious that some rows of W would have all zero entries which correspond to those features that are not relevant to any projects.

3 | OUR PROPOSED MASK APPROACH

Previous approaches[38,22] can build one defect prediction model for one target project at a time, while MASK extended from a previous study[31] can build multiple defect prediction models for multiple target projects at a time. Figure. 2 presents the overall framework of MASK. In this figure, for the convenience to distinguish related projects, these projects are represented with different colors respectively. Besides, these projects are drawn in the form of three-dimensions, which can easily display the format of input data. In the phase of model building, different colors are used to represent different kinds of learned information (i.e., shared-information and non-shared-information), in which α and β are regularization parameters and their detailed introduction can be found in Section 3.1. In general, MASK mainly contains two phases: a differential evolution optimization (DEO) phase and a multi-task learning (MTL) phase. In the former phase, since hyperparameter optimization[43] has a substantial impact on the model performance and cannot be neglected, we use DEO to search for the optimal value of regularization parameters for both the shared information and non-shared information. In the latter phase, to address the issue of insufficient labeled modules, we use multi-task learning to build models simultaneously, based on the optimal value of regularization parameters obtained by DEO. Finally, we return models for different projects.

Different from previous studies which take each project as input independently, our framework takes all related projects as a whole as the input and it is shown in Figure. 2. Some data preprocessing methods are performed on the datasets of these projects. In particular, we normalize all the numerical features by using z-score. Here the number of related projects to be learned equals to the number of models outputted by MASK. Relatedness is a core concept in multi-task learning. In this paper, we simply consider two types of relatedness. One is whether these projects are developed by the same organization. The other is whether these projects have the same feature space. A more effective way to measure the relatedness of different projects can help improve the performance of Mask. This is still an open problem and needs more investigation in the future.

In the rest of this section, we will introduce the details of differential evolution optimization phase and multi-task learning phase in sequence.

3.1 | Differential Evolution Optimization Phase

In this phase, MASK uses differential evolution optimization to search for the optimal hyperparameter value for both the shared information and the non-shared information. DEO is an optimizer, which is especially suitable for functions that may not be smooth or linear. It has been widely used in previous studies for hyperparameter optimization[44,45,46,47]. According to a previous study[48], DEO has more competitive advantages when compared with other meta-heuristic search algorithms, such as particle swarm optimization or genetic algorithm.

Algorithm 1 Differential Evolution Optimization Phase

Input:

$X^{n\times F\times t}$: all the data of related projects
n: population size
p: crossover probability
w: differential weight
$lives$: number of generations

Output:

$BRP(\alpha, \beta)$: a pair of the optimal regularization parameters;

1: Split X into two sets: training set written as X_T and validation set written as X_V;
2: Let $candicatePop$ = Initial population with n chromosomes;
3: Build models on X_T with $candicatePop$ and evaluate by AUC measure on X_V. Then record the best solution (i.e., the solution with maximum AUC) found so far as BRP;
4: Let $lives = 1$;
5: **while** $(lives-- > 0)$ **do**
6: **for** $i = 1$ to n **do**
7: P_i= the i-th chromosome of $candicatePop$;
8: NP = Mutation(P_i,p,w);
9: NP = Crossover(NP, P_i,p,w);
10: Compare the fitness of both NP and P_i on X_V by $func$, and record the better generation;
11: **if** (the better generation is greater than BRP) **then**
12: Update BRP and allow further iteration: $lives$++;
13: **end if**
14: **end for**
15: **end while**
16: **return** BRP

In Algorithm 1, DEO firstly splits all related projects X into two sets: a training set written as X_T and a validation set written as X_V (Line 1). Then, it randomly initializes the value of each candidate chromosome which encodes two regularization parameters (Line 2). After that, these models built on X_T with n candidates chromosomes from an initial population are evaluated for the current regularization parameters on the validation data X_V. Notice that the evolution is guided by a fitness function (i.e., maximizing AUC calculated by $func$ in Line 10) (Line 3). It starts to evolve on the initial population with one $lives$ (Line 4). After that, DEO goes into population evolution[48](Lines 5-15). In each generation, to evolve the candidates, DEO compares the fitness value (i.e., AUC) of old candidates with that of the new candidates. After that, new candidates with better fitness value replace older ones (Line 10). In particular, the $func$ function returns the average performance of MTL (Algorithm. 2) using the evolved new parameter settings. Notice that we initialize parameter $lives$ with 1, which can ensure it runs into the inner of $while$ loop. Besides, there also exists a for loop inside of the $while$ loop. In the for loop, the differential evolution algorithm will increase the value of parameter $lives$ if better generation can be found (Lines 11-12). Therefore, there is at least one generation if differential evolution cannot find better generations. When the DEO optimizer terminates, it will return the optimal value from the candidate parameter setting BRP (i.e., two regularization parameters) obtained in all the iterations.

TABLE 1 Parameters in the DEO Phase

Parameter Name	Description	Default Value
n	frontier size in a generation	10
p	survival of the candidate	0.3
w	mutation probability	0.7
$lives$	number of generations	1

Table 1 and Table 2 list all the important parameters in the DEO phase and the regularization parameters. Table 1 provides parameter names, their descriptions and default values in the DEO phase. In this paper, we use the value of hyperparameters in differential evolution, which is suggested by Agrawal et al.[47]. Table 2 provides regularization parameter names, search ranges explored by DEO and their descriptions. Notice we use the tuning range of these regularization parameters suggested by Jalali et al.[31].

TABLE 2 Regularization Parameters in the MTL Phase

Parameter Name	Tuning Range (Explored by DEO)	Description
α	[0.01, 2]	a regularization parameter for shared-information
β	[0.01, 2]	a regularization parameter for non-shared information

3.2 | Multi-Task Learning Phase

Algorithm 2 presents the details on the multi-task learning phase. MTL firstly needs to initialize the start points. In our implementation, we simply initialize the start points with zero matrices (Line 1). In fact, a starting point can be initialized to a value computed from data or a specified point. It defines some important variables which can control the process of searching for the optimal value for S, NS and B (Line 2). Then, MTL goes into a loop block with $maxIter$ iteration times (Lines 3-15). In this loop, MTL updates $Shared$, $NonShared$ and $Bias$ with the search factor α (Line 4). After that, it computes the whole loss value and gradients for the current search points (Line 5). Next, MTL goes further into a inner loop in which it can only be terminated until a specific termination condition is satisfied (Lines 6-12). In this inner loop, the same normalization operations[31] are used on the shared and non-shared information parameters together with the optimal bias data, and the Armijo Goldstein line search scheme[49] is used to speed up search efficiency (Lines 7-9). MTL then calculates the performance of current parameters and records the performance (Line 10). When satisfying the termination condition of the inner loop, it backs up previous search points, does preparation for the next iteration and jumps out of the inner loop block (Line 11). If the difference between the latest two performance values meets the requirements of maximum tolerance, MTL can jump out of the outer loop block

Algorithm 2 Multiple Tasks Learning Phase

Input:

$X^{n\times F\times t}$: all the data of related projects
$Y^{n\times 1\times t}$: the corresponding true labels of all related projects
BRP: the output of DEO
$maxIter$: the number of maximum iterations
$maxTol$: the tolerance of the performance between two iteration

Output:

$Shared^{F\times t}(S)$: all shared information among projects
$NonShared^{F\times t}(NS)$: non-shared information for each project
$Bias^{t\times 1}(B)$: the bias data for classification

1: Initialization the start point: filling $Shared$, $NonShared$, $Bias$ with a zero matrix;
2: Let t=1, $t' = iter = 0$, $\gamma = 1, \gamma_{inc} = 2$, $\alpha =$ Initial search factor, which controls the search for optimal S, NS, B;
3: **while** $(iter < maxIter)$ **do**
4: Set $\alpha = (t' - 1)/t$, then find next search point for S, NS, B with α, and store them into S_{tmp}, NS_{tmp}, B_{tmp};
5: Compute the whole loss value with regularization parameters stored in BRP and gradients of the current search point with S_{tmp}, NS_{tmp}, and B_{tmp};
6: **while** $(True)$ **do**
7: process proximal $l1 - norm$ and $l\infty - norm$ on S_{tmp} in turn, and return S_t;
8: process proximal $l_1 - norm$ and $l_1 - norm$ on NS_{tmp} in turn, and return NS_t;
9: Update bias vector to B_t;
10: Evaluate performance and save it in list of $funcVal$;
11: Stop if reach termination conditions, otherwise let $\gamma = \gamma \times \gamma_{inc}$;
12: **end while**
13: If reach maximum tolerance, **Break**;
14: Let $iter$++, and update t' and t;
15: **end while**
16: **return** S, NS, B

and end in advance (Line 13). If not, MTL will go to the next iteration with the updated change factor (i.e., $t^{'}$, t) (Line 14). At last, it returns the optimal value for S, NS and B calculated by a certain number of iterations (Line 16).

3.3 | Working Example of MASK

To facilitate the understanding of the process of MASK, we take a working example to illustrate how MASK works in Figure. 3. Suppose there are three related projects (i.e., A, B and C) with limited labeled data, and all of these projects need defect prediction. However, we can use neither WPDP methods nor CPDP methods to build prediction models since the limitation of labeled data from source project or target projects. Therefore, we resort to MASK since it does well in such a situation that each project has a few labeled data. All three projects will be treated as the input of MASK simultaneously. MASK firstly splits the three projects into two sets respectively: a training set X and a testing set Y. To obtain the optimal value for hyperparameters, MASK further split X into two sets: training dataset X_T and validation dataset X_V. In this step, DEO, an optimizer, searches for the optimal value for hyperparameters on X_T and checks the validation on X_V. After DEO obtains the optimal value for hyperparameters, MASK then builds three prediction models simultaneously and predict the testing data Y respectively.

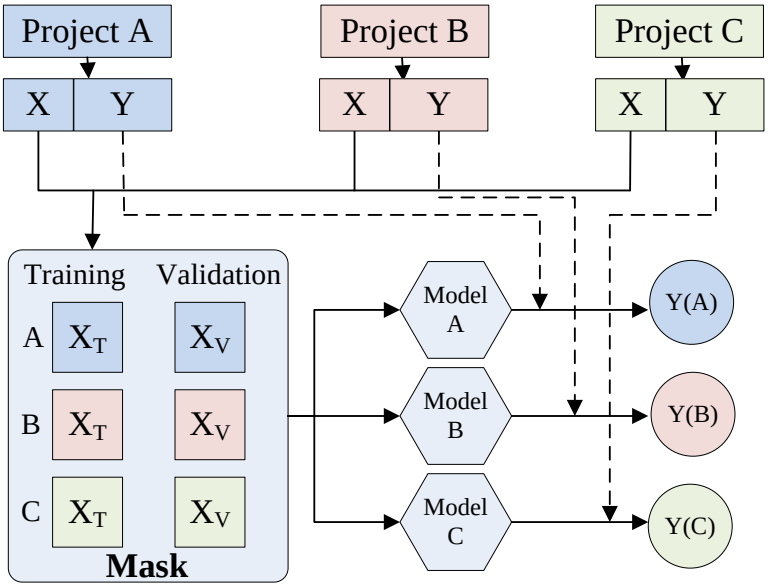

FIGURE 3 A Working Example When Applying MASK to Three Related Projects.

4 | EXPERIMENT DESIGN AND RESULT ANALYSIS

In this section, we conduct empirical studies to evaluate the performance of MASK. All the approaches are executed on CentOS Linux 7 (Memory: 32GB, Processor: Intel Xeon(R) CPU E5-2609 v2 @ 2.50GHz × 8).

4.1 | Experimental Setup

In our empirical studies, we assume that both the source and target projects have limited labeled data in our proposed new scenario. To solve the issue in this scenario, MASK based on multi-task learning is used. Here, we treat one project in the context of SDP as one task in the context of MTL.

To evaluate the performance of MASK, we consider four baseline approaches. The first baseline approach is single-task learning (STL). This approach treats each project independently. The second baseline approach is proposed by Zimmermann et al.[27]. This approach directly combines all related project and we use SCL (simple combined learning) to represent this approach. The third baseline approach is Peters filter proposed by Peters et al.[37]. The forth baseline approach is Burak filter proposed by Turhan et al.[16]. The last two approaches are classical module selection based CPDP approaches. Notice that we do not take TCA+[39] as the baseline method since it needs no labeled modules in the target project to build a prediction model but MASK does need. In addition, multi-label classification techniques[50] have been adopted into the software engineering domain. However, there also exists a difference between multi-task learning techniques and multi-label classification techniques. In particular, multi-label classification techniques aim to deal with these issues in which modules must have at least three different types of label, while multi-task classification techniques aim to build multiple defect prediction models simultaneously which is suitable

for both multi-label classification issues and binary classification issues. However, for defect prediction considered in this paper, all the experimental datasets (i.e., AEEEM, MORPH and ReLink) have at most two different types of the label (i.e., defectiveness or non-defectiveness). Therefore, multi-label classification techniques are not suitable for these datasets considered in the current experiment, and we do not take such techniques as the baseline methods for comparison.

For MASK, we train and test the defect prediction models on all projects simultaneously. Since we want to simulate the insufficiency of training data for a newly developed project, we use inverse 10-fold cross validation (CV). In particular, for a specific project (i.e., the target project), we split the project into ten folds equally by using stratified sampling. Then, the one fold is treated as the labeled training data, while the remaining nine folds are treated as the testing data. For example, there are three related projects: A, B and C. All of them are treated as the target projects since we want to do defect prediction on them independently. We first split these three datasets into ten folds respectively. Then the labeled training data is comprised of the i-th fold of project A, the i-th fold of project B, and the i-th fold of project C (i.e., $i \in [1,10]$). Notice these folds are not simply combined together. After that, MASK builds three prediction models for these projects simultaneously. When predicting on project A, the prediction model built by MASK for project A is tested on the remaining nine folds of the project A. This process is repeated 10 times and we report performance value of 10 runs on average for the project A (i.e., AVG is an abbreviation for average in Figure. 4). The main evaluation process of MASK can be found in Figure. 4.

For STL, we also use inverse 10-fold CV to train and test defect prediction model for each project independently. For example, when performing defect prediction on the project A, STL only builds the prediction model on one fold of the project A and evaluates on the remaining nine folds. This process is repeated 10 times and we report the average performance value of 10 runs for the project A.

For three CPDP methods (i.e., SCL, Peters filter, and Burak filter), inverse 10-fold CV is performed differently. For example, there are three projects (i.e., A, B, and C). We first split these three datasets into ten folds respectively. Then these CPDP methods combine the i-th fold of the project A, the i-th fold of the project B, and the i-th fold of the project C as the labeled training data. Notice these folds are only simply combined together. After that, a prediction model built on the combined three folds is evaluated on the remaining nine folds of each project respectively. This process is repeated 10 times and we report the average performance value of 10 runs for three projects respectively.

To overcome a possible bias in the data split processes, we perform inverse 10-fold CV for 10 times independently.

4.2 | Datasets

In the experiments, we use three widely used datasets [39,22,24,51,38]: ReLink, AEEEM and MORPH. Table 3 shows the characteristics of these datasets. The ReLink dataset was collected by Wu et al. [52]. ReLink considers 26 complexity metrics as features. The metrics are designed based on the analysis on code complexity (such as LOC, cyclomatic complexity, number of classes, etc.),

TABLE 3 The Characteristics of Datasets.

Datasets	Projects	# Metrics	# Modules	#(%) defective Modules
ReLink	Apache	26	194	98 (50.52%)
	Safe		56	22 (39.29%)
	ZXing		399	118 (29.57%)
AEEEM	EQ	61	324	129 (39.8%)
	JDT		997	206 (20.7%)
	LC		691	64 (9.3%)
	ML		1,862	245 (13.2%)
	PDE		1,497	209 (14.0%)
MORPH	ant-1.3	20	125	20 (16.00%)
	arc		234	27 (11.54%)
	camel-1.0		339	13 (3.83%)
	poi-1.5		237	141 (59.49%)
	redaktor		176	27 (15.34%)
	skarbonka		45	9 (20.00%)
	tomcat		858	77 (8.97%)
	velocity-1.4		196	147 (75.00%)
	valan-2.4		723	110 (15.21%)
	xerces-1.2		440	71 (16.14%)

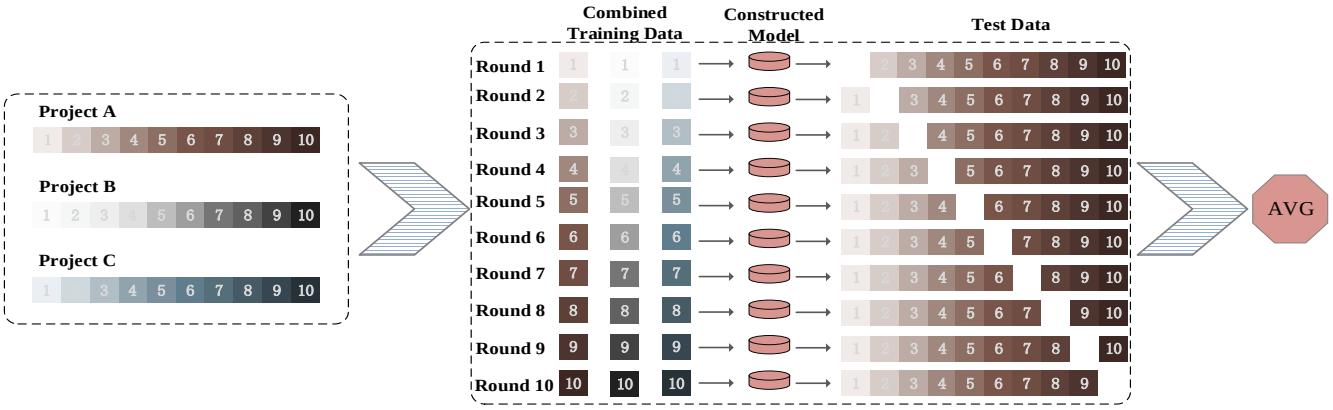

FIGURE 4 The Evaluation Process of Inverse 10-Fold Cross Validation.

and abstract syntax trees (such as number of blocks, number of statements, method references, etc.). The AEEEM dataset was collected by D'Ambros et al.[53]. AEEEM considers 61 metrics, which are categorized into source code metrics, previous-defect metrics, entropy-change metrics, and churn-of-source-code metrics. The MORPH dataset can be downloaded from PROMISE repository[‡] , which considers CK based metrics. Notice that in this paper, we assume these projects in the same dataset are developed by the same organization. Besides, the projects in the same dataset use the same metrics to measure the extracted program modules (i.e., they have the same feature space). Therefore, in terms of these two heuristics (i.e., the same organization

[‡]http://openscience.us/repo

and the same feature space), we assume all projects in the same dataset have a certain relatedness. For example, $Apache$, $Safe$, and $ZXing$ are considered as related project since all of them belong to ReLink data set and they have the same feature space (i.e., 26 metrics).

4.3 | Performance Measures

We use F1 and AUC to evaluate the performance of different approaches. According to the ground truths and the predicted class labels, we can compute the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) respectively.

F1 measure, a trade-off between the precision measure (P) and the recall measure (R), is widely used to compare different defect prediction approaches[22] and F1 can be computed as follows:

$$F1 = (2 \times P \times R)/(P + R). \tag{1}$$

Here the measures P and R can be computed as follows:

$$P = TP/(TP + FP),\ R = TP/(TP + FN). \tag{2}$$

Different from F1, AUC measures the area under the receiver operating characteristic (ROC) curve, which is a 2D illustration of true positive rate on the y-axis versus false positive rate on the x-axis. ROC curve is obtained by varying the classification threshold over all possible values, separating non-defective and defective predictions. A prediction model with the best performance provides an AUC value close to 1. The ROC analysis is robust in case of imbalanced class distributions and asymmetric misclassification costs[43,54,55,56,57].

4.4 | Result Analysis

In our empirical studies, we want to answer the following two research questions.

RQ1: *How effective is our proposed approach* MASK *in defect prediction? How much improvement can it achieve over the baseline approaches?*

Motivation. Our goal is to design an approach , which is applicable to new usage scenario that both source and target projects have limited labeled modules. However, to show the feasibility of this approach, this RQ is designed to find how effective it is in performing defect prediction and whether it can have better performance than baseline approaches. The answer for this RQ would shed light on how much our approach advances the state-of-the-art approaches in the defect prediction.

TABLE 4 Cliff's Delta and the Effectiveness Level[58].

Cliff's Delta(δ)	Effectiveness Level
$\|\delta\| < 0.147$	Negligible
$0.147 \leq \|\delta\| < 0.33$	Small
$0.33 \leq \|\delta\| < 0.474$	Medium
$\|\delta\| \geq 0.474$	Large

Approach. To answer this RQ, we compare MASK with STL, SCL, Peters filter and Burak filter to investigate whether MASK has better performance over these baseline approaches for building defect prediction models. F1 and AUC are used to evaluate the performance of these approaches. By default, we run all these approaches 100 times independently. To check the significance of performance comparison, we conduct the Wilcoxon signed-rank test[59] with a Bonferroni correction[60]. Wilcoxon signed-rank test is a non-parametric statistical hypothesis test on the performance measures, while Bonferroni correction is used to counteract the problem of multiple comparisons. For all the statistical testings, the null hypothesis is that there is no difference between the trained prediction models, and the significance level α is set as 0.05. If p-value is smaller than 0.05, we reject the null hypotheses; otherwise we accept the null hypotheses.

We also use Cliff's delta (δ)[58], which is a non-parametric effect size measure and can quantify the amount of difference between two approaches. In our context, the Cliff's delta is used to compare our proposed approach MASK with the baseline approaches. The range of Cliff's delta is [-1,1]. If the absolute value of δ equals to 1, it indicates the absence of overlap between two approaches. It means all data from one group are higher than that from the other group, and vice versa. If the value of δ equals to zero, it means that the two approaches are overlapping completely. Table 4 describes the meanings and corresponding interpretations when Cliff's delta has different values[58].

Results. Table 5 and Table 6 show the comparison results of MASK with STL and the state-of-the-art CPDP approaches (i.e., SCL, Peters filter and Burak filter) in terms of F1 and AUC, respectively. In both of the two tables, the first column presents the name of datasets. The second column lists project name. Here we assume that the projects in the same dataset has a certain relatedness, since they consider the same metrics. The remaining 5 columns list the performance values of five approaches in terms of different performance measures (i.e., F1 and AUC). The results in cells are in the form of mean $\pm$ standard deviation, which can illustrate the average performance and stability of trained models. Notice that for these baseline approaches, we also use Logistic regression as the underlying classifier, since the underlying classifier used by MASK is Logistic regression. This setting can guarantee that these methods are compared in a fair way.

We further analyze whether the improvement of MASK over baseline approaches are significant based on Wilcoxon signed-rank test[59] with a Bonferroni correction[60] for each project. The detailed results are shown in Table 7. If the statistical test shows a significant difference, we then use the Cliff's δ to examine whether the magnitude of the difference is not negligible. In Table

TABLE 5 The Comparison Result of MASK with STL, SCL, Peters Filter and Burak Filter in Terms of F1. Notice the Best F1 in Each Row Is Marked in Bold.

Datasets	Related Projects	MASK	STL	SCL	Peters Filter	Burak Filter
AEEEM	EQ	**0.574**±0.061	0.529±0.058	0.322±0.048	0.511±0.117	0.545±0.134
	JDT	**0.539**±0.090	0.404±0.057	0.428±0.053	0.320±0.105	0.452±0.143
	LC	**0.365**±0.071	0.222±0.052	0.302±0.062	0.145±0.081	0.299±0.094
	ML	0.255±0.102	**0.297**±0.034	0.276±0.035	0.231±0.069	0.291±0.065
	PDE	0.284±0.100	0.277±0.037	**0.295**±0.039	0.188±0.112	**0.295**±0.113
Avg		**0.403**±0.085	0.346±0.048	0.325±0.047	0.279±0.096	0.376±0.110
MORPH	ant-1.3	**0.338**±0.137	0.289±0.105	0.199±0.116	0.284±0.109	0.252±0.142
	arc	**0.270**±0.111	0.208±0.073	0.232±0.104	0.166±0.093	0.063±0.101
	camel-1.0	**0.185**±0.080	0.145±0.063	0.138±0.067	0.080±0.040	0.056±0.071
	poi-1.5	**0.733**±0.037	0.587±0.060	0.089±0.035	0.428±0.342	0.317±0.164
	redaktor	**0.360**±0.140	0.298±0.117	0.167±0.097	0.222±0.111	0.202±0.173
	skarbonka	0.327±0.136	**0.337**±0.115	0.208±0.088	0.223±0.164	0.187±0.160
	tomcat	0.256±0.102	0.247±0.060	**0.300**±0.068	0.122±0.090	0.139±0.093
	velocity	**0.858**±0.038	0.726±0.091	0.067±0.043	0.558±0.359	0.079±0.057
	xalan-2.4	0.279±0.106	0.294±0.055	0.205±0.070	0.166±0.124	**0.333**±0.055
	xerces-1.2	0.138±0.059	**0.291**±0.057	0.124±0.052	0.216±0.085	0.204±0.042
Avg		**0.374**±0.095	0.342±0.080	0.173±0.074	0.247±0.152	0.183±0.106
Relink	Apache	**0.617**±0.052	0.507±0.087	0.544±0.079	0.612±0.084	0.611±0.066
	Safe	**0.545**±0.141	0.496±0.148	0.504±0.117	0.493±0.209	0.543±0.112
	Zxing	0.225±0.109	**0.395**±0.050	0.390±0.056	0.378±0.160	0.393±0.057
Avg		0.462±0.101	0.466±0.095	0.479±0.084	0.494±0.151	**0.516**±0.078
Total Avg		**0.397**±0.093	0.364±0.073	0.266±0.068	0.297±0.136	0.292±0.102

TABLE 6 The Comparison Result of MASK with STL, SCL, Peters Filter and Burak Filter in Terms of AUC. Notice the Best AUC in Each Row Is Marked in Bold.

Datasets	Related Projects	MASK	STL	SCL	Peters Filter	Burak Filter
AEEEM	EQ	**0.677**±0.027	0.619±0.037	0.581±0.018	0.621±0.163	0.581±0.061
	JDT	**0.709**±0.052	0.624±0.041	0.638±0.025	0.561±0.073	0.630±0.058
	LC	**0.657**±0.054	0.582±0.045	0.622±0.039	0.528±0.083	0.577±0.072
	ML	0.580±0.046	**0.601**±0.025	0.583±0.017	0.560±0.058	0.559±0.051
	PDE	**0.592**±0.043	0.580±0.026	0.589±0.019	0.526±0.071	0.554±0.052
Avg		**0.643**±0.044	0.601±0.035	0.603±0.024	0.559±0.090	0.580±0.059
MORPH	ant-1.3	0.606±0.094	0.563±0.091	0.542±0.049	**0.643**±0.090	0.555±0.108
	arc	**0.574**±0.070	0.542±0.069	0.565±0.046	0.543±0.099	0.538±0.093
	camel-1.0	0.553±0.072	**0.556**±0.061	0.534±0.047	0.539±0.090	0.544±0.090
	poi-1.5	**0.643**±0.047	0.536±0.053	0.517±0.010	0.490±0.091	0.605±0.059
	redaktor	**0.606**±0.086	0.573±0.105	0.517±0.046	0.507±0.093	0.535±0.090
	skarbonka	0.563±0.119	**0.573**±0.106	0.494±0.044	0.482±0.101	0.505±0.107
	tomcat	0.595±0.058	0.595±0.041	**0.611**±0.038	0.508±0.087	0.555±0.086
	velocity	**0.661**±0.075	0.596±0.086	0.483±0.026	0.500±0.044	0.574±0.094
	xalan-2.4	**0.585**±0.045	0.584±0.037	0.553±0.025	0.481±0.074	0.571±0.067
	xerces-1.2	0.517±0.019	**0.575**±0.042	0.523±0.016	0.493±0.036	0.532±0.052
Avg		**0.590**±0.069	0.569±0.069	0.534±0.035	0.519±0.080	0.551±0.085
Relink	Apache	**0.643**±0.068	0.521±0.067	0.571±0.070	0.629±0.068	0.621±0.070
	Safe	**0.648**±0.092	0.574±0.126	0.589±0.101	0.558±0.138	0.627±0.110
	Zxing	0.528±0.025	**0.554**±0.036	0.553±0.041	0.512±0.063	0.550±0.044
Avg		**0.606**±0.062	0.550±0.076	0.571±0.071	0.566±0.090	0.599±0.075
Total Avg		**0.608**±0.061	0.575±0.061	0.559±0.038	0.538±0.085	0.567±0.076

7, the cells in "*p*-value" column filled with light blue indicates MASK performs significantly better than baseline approaches according to the Wilcoxon signed-rank test with a Bonferroni correction (i.e., *p*-value is less than 0.05). Notice that there are 10×10 results for each comparison between MASK and baselines approaches on a specific project. When MASK is statistically better than baselines, the cells in δ column will be filled with dark grey or light grey based on whether the magnitude of the difference is trivial according to Cliff's δ. (i.e., filling light grey if effectiveness level is negligible or dark grep if effectiveness level is non-negligible).

Compared with STL, MASK can achieve better performance across 18 projects in most cases. In terms of F1, MASK achieves average performances of 0.403, 0.374 and 0.462 on AEEEM, MORPH and ReLink, respectively. MASK has better performance than STL on AEEEM and MORPH. The reason of worse performance on ReLink may arise from the relatively small number of related projects (i.e. only three related projects). In terms of AUC, MASK achieves average performances of 0.643, 0.590 and 0.606 on AEEEM, MORPH and ReLink, respectively. MASK has better performance than STL on all the datasets on average and has 23.42% improvement over STL at most. According to Table 7, we notice that in most cases, MASK shows significant improvement over the STL with non-negligible effect size in terms of both F1 and AUC performance measures.

Compared with SCL, MASK can achieve better performance on average. In terms of F1, MASK achieves performances of 0.403, 0.374 and 0.462 on AEEEM, MORPH and ReLink, respectively. MASK has better performance than SCL except for ReLink on average and has 49.25% improvement over SCL. In terms of AUC, MASK performs better than SCL in most cases and has 36.85% improvement at most. According to Table 7, we notice that in most cases, MASK shows significant improvement over SCL and has non-negligible effect size when considering both F1 and AUC performance measures.

Compared with Peters filter, MASK can achieve better performance on average. In terms of F1, MASK achieves performances of 0.403, 0.374 and 0.462 on AEEEM, MORPH and ReLink, respectively. MASK has better performance than Peters filter except for ReLink on average and has 33.67% improvement over Peters filter. In terms of AUC, MASK performs better than Peters filter for each project and improves Peters filter by 32.20% of AUC at most. According to Table 7, we notice that in most cases, MASK shows significant improvement over Peters filter and has non-negligible effect size when considering both F1 and AUC performance measures.

Compared with Burak filter, MASK still achieves better performance on average. In terms of F1, MASK achieves performances of 0.403, 0.374 and 0.462 on AEEEM, MORPH and ReLink, respectively. MASK has better performance than Burak filter except for ReLink on average and has 35.96% improvement over Burak filter. In terms of AUC, MASK performs better than Burak filter for nearly all the projects. The results of MASK on average are better than Burak filter across 18 projects and have 16.52% improvement over Burak filter at most. According to Table 7, we notice that in most cases, MASK shows significant improvement over Burak filter and has non-negligible effect size when considering both F1 and AUC performance measures.

TABLE 7 *P*-Value and Cliff's Delta (δ) for Comparison Results Between MASK and the Baseline Approaches in Terms of F1 and AUC.

Datasets	Projects	MASK vs. STL				MASK vs. SCL				MASK vs. Peter Filter				MASK vs. Burak Filter			
		F1		AUC		F1		AUC		F1		AUC		F1		AUC	
		p-value	δ	*p*-value	δ	*p*-value	δ	*p*-value	δ	*p*-value	δ	*p*-value	δ	*p*-value	δ	*p*-value	δ
AEEEM	EQ	2.39E-08	0.46	2.23E-22	0.80	2.65E-33	0.98	1.29E-33	0.99	5.36E-04	0.28	2.42E-03	0.25	1.55E-01	-0.12	8.24E-24	0.82
	JDT	2.39E-22	0.80	2.50E-21	0.78	1.08E-19	0.74	2.15E-19	0.74	2.72E-28	0.90	9.64E-29	0.91	1.34E-11	0.55	1.83E-17	0.70
	LC	4.44E-26	0.87	1.36E-17	0.70	9.30E-12	0.56	1.06E-07	0.44	7.36E-31	0.95	4.10E-22	0.79	8.00E-11	0.53	1.02E-14	0.63
	ML	6.24E-02	-0.15	1.36E-03	-0.26	5.43E-01	-0.05	4.10E-01	-0.07	6.07E-02	0.15	1.31E-03	0.26	1.24E-01	-0.13	6.32E-03	0.22
	PDE	9.44E-02	0.14	3.95E-02	0.17	5.52E-01	0.05	5.99E-01	0.04	3.66E-08	0.45	4.54E-13	0.59	2.75E-01	-0.09	3.47E-07	0.42
MORPH	ant-1.3	6.01E-03	0.22	4.90E-03	0.23	8.09E-11	0.35	1.41E-06	0.39	3.40E-03	0.25	2.66E-03	-0.25	1.02E-04	0.33	3.95E-03	0.24
	arc	3.65E-05	0.36	2.09E-03	0.25	2.00E-02	0.12	4.07E-01	0.07	1.24E-08	0.49	2.27E-02	0.19	7.54E-22	0.81	3.19E-03	0.24
	camel-1.0	5.95E-03	0.10	1.63E-01	-0.11	4.63E-04	-0.05	1.47E-01	0.12	1.31E-16	0.80	4.01E-01	0.07	7.88E-16	0.76	6.90E-01	-0.03
	poi-1.5	2.84E-31	0.95	2.35E-25	0.85	2.48E-34	1.00	1.11E-31	0.96	2.22E-06	0.39	4.39E-26	0.86	4.95E-34	1.00	2.70E-07	0.42
	redaktor	1.72E-04	0.32	1.20E-02	0.21	1.35E-13	0.20	4.32E-12	0.57	6.61E-12	0.59	2.53E-10	0.52	2.16E-09	0.51	2.11E-07	0.42
	skarbonka	4.75E-01	-0.17	3.61E-01	-0.07	7.55E-06	-0.47	6.73E-08	0.44	3.04E-04	0.30	3.22E-07	0.42	2.82E-08	0.47	2.47E-04	0.30
	tomcat	7.53E-02	0.15	6.36E-01	-0.04	8.80E-03	-0.22	7.63E-03	-0.22	4.83E-16	0.67	1.94E-15	0.65	2.13E-13	0.60	2.13E-04	0.30
	velocity	2.01E-24	0.83	4.17E-08	0.45	8.87E-33	0.82	1.70E-32	0.97	1.38E-17	0.70	8.94E-29	0.91	2.55E-34	1.00	5.14E-11	0.54
	xalan-2.4	9.75E-01	0.00	7.63E-01	0.02	4.69E-09	0.48	1.50E-08	0.46	5.14E-11	0.54	7.03E-25	0.84	4.83E-04	-0.29	9.88E-02	0.14
	xerces-1.2	8.87E-28	-0.93	3.42E-21	-0.77	6.07E-02	0.15	1.56E-02	-0.20	5.53E-13	-0.62	7.67E-07	0.40	3.90E-14	-0.65	2.93E-03	-0.24
Relink	Apache	1.89E-17	0.70	1.12E-22	0.80	5.37E-12	0.57	2.87E-14	0.62	7.77E-01	-0.02	1.58E-02	0.20	5.02E-01	0.06	7.06E-03	0.22
	Safe	2.05E-03	0.25	1.88E-06	0.39	1.66E-04	0.31	1.08E-06	0.40	1.60E-01	0.12	1.62E-08	0.46	1.82E-01	0.11	1.09E-01	0.13
	Zxing	1.02E-24	-0.89	8.62E-08	-0.44	3.35E-23	-0.86	2.37E-07	-0.42	6.97E-14	-0.65	2.30E-01	0.10	4.04E-23	-0.86	3.96E-04	-0.29

In this RQ, we assume only 10% of modules are labeled for each project. The number of 10% modules in Relink, AEEEM, and MORPH varies from 19∼40, 32∼150, and, 13∼72 respectively. Therefore, it is not hard to find that our MASK only requires a small amount of labeled data and it can achieve competitive performance based on the above result analysis.

In addition, STL only uses limited data from the target project, and SCL simply combines the labeled data from other related projects. Both of them (i.e., STL and SCL) ignore the relatedness among all related projects. However, MASK builds multiple prediction models simultaneously by utilizing the relatedness among all related projects. By comparing the performance value with STL and SCL, we can find that MASK can outperform these two methods significantly, which indicates, to some extent, MASK can mine the relatedness among related projects successfully and then improve the prediction performance.

For a new usage scenario that both the source and target projects have limited labeled data, our proposed approach MASK*, which builds prediction models for related projects simultaneously, has better performance than state-of-the-art baseline approaches (i.e., STL, SCL, Peters filter and Burak filter). These promising results show that* MASK *can effectively extract and utilize shared information in related projects when building prediction models.*

RQ2: *How does the number of labeled modules affect the performance of* MASK*?*

Motivation. Our proposed new usage scenario has the issue that both the source project and the target projects have limited labeled modules. In RQ1, we use inverse 10-fold CV to simulate the new scenario with insufficient labeled modules. That is, by default, we only set the percentage of labeled modules as 10%. But the performance of our approach may vary when the percentage of the labeled modules is different. Therefore, we want to investigate the influence of different percentages of labeled modules on the performance of MASK.

Approach. To answer this RQ, we perform two additional experiments on another two settings: inverse 5-fold CV and inverse 2-fold CV. That is, we will investigate the performance of MASK when the percentage of labeled trained data is set as 10%, 20% and 50%, respectively.

Results. Table 8 presents the average F1 and AUC across the 18 projects with various percentages of labeled modules as the training data. In Table 8, the first column presents the name of datasets. The second column lists related projects for each datasets. The following six columns are divided into two groups for F1 and AUC performance measures. In each group, there are three settings representing the percentages of labeled modules for MASK, which can illustrate the average performance and stability of trained models. The average values of 100 runs are listed at the end of each dataset.

In terms of F1, the average values vary from 0.138∼0.858, 0.107∼0.873 and 0.103∼0.883 by using 10%, 20% and 50% modules which are labeled. With the increase in the number of labeled modules, the performance of MASK is improved. For example, MASK on average can achieve 0.397, 0.407 and 0.427 by using 10%, 20% and 50% modules which are labeled.

TABLE 8 The Performance of MASK When Considering Different Percentage of Labeled Modules in Terms of F1 and AUC. Notice the Best F1 and AUC Are Marked in Bold.

Datasets	Related Projects	F1			AUC		
		10%	20%	50%	10%	20%	50%
AEEEM	EQ	0.574±0.061	0.574±0.074	**0.597**±0.057	0.677±0.027	0.678±0.034	**0.688**±0.029
	JDT	0.539±0.090	0.540±0.097	**0.564**±0.062	0.709±0.052	0.708±0.052	**0.717**±0.037
	LC	0.365±0.071	0.378±0.078	**0.406**±0.062	0.657±0.054	0.658±0.052	**0.665**±0.044
	ML	0.255±0.102	0.243±0.093	**0.245**±0.097	**0.580**±0.046	0.574±0.035	0.575±0.041
	PDE	0.284±0.100	0.282±0.112	**0.302**±0.094	0.592±0.043	0.590±0.047	**0.598**±0.044
Avg		0.403±0.085	0.403±0.091	**0.423**±0.075	0.643±0.044	0.642±0.044	**0.649**±0.039
MORPH	ant-1.3	0.338±0.137	0.350±0.132	**0.391**±0.123	0.606±0.094	0.626±0.085	**0.631**±0.098
	arc	0.270±0.111	**0.285**±0.121	0.256±0.111	0.574±0.070	**0.587**±0.070	0.567±0.063
	camel-1.0	0.185±0.080	**0.211**±0.067	0.200±0.032	0.553±0.072	**0.560**±0.070	0.510±0.037
	poi-1.5	0.733±0.037	0.749±0.024	**0.760**±0.035	0.643±0.047	0.667±0.026	**0.679**±0.030
	redaktor	0.360±0.140	0.415±0.113	**0.529**±0.089	0.606±0.086	0.638±0.078	**0.705**±0.050
	skarbonka	0.327±0.136	0.346±0.123	**0.361**±0.103	0.563±0.119	**0.581**±0.100	0.574±0.090
	tomcat	0.256±0.102	0.288±0.081	**0.319**±0.069	0.595±0.058	0.608±0.042	**0.615**±0.037
	velocity	0.858±0.038	0.873±0.034	**0.883**±0.031	0.661±0.075	0.711±0.042	**0.734**±0.042
	xalan-2.4	0.279±0.106	0.297±0.087	**0.327**±0.070	0.585±0.045	0.592±0.037	**0.603**±0.034
	xerces-1.2	**0.138**±0.059	0.107±0.050	0.103±0.056	**0.517**±0.019	0.512±0.014	0.510±0.017
Avg		0.375±0.095	0.392±0.083	**0.413**±0.072	0.590±0.069	0.608±0.056	**0.613**±0.050
Relink	Apache	0.617±0.052	**0.630**±0.044	0.628±0.058	0.643±0.068	0.668±0.038	**0.689**±0.033
	Safe	0.545±0.141	0.559±0.128	**0.602**±0.095	0.648±0.092	0.672±0.061	**0.691**±0.068
	Zxing	0.225±0.109	**0.227**±0.111	0.215±0.089	0.528±0.025	0.532±0.026	**0.533**±0.018
Avg		0.462±0.101	0.472±0.094	**0.482**±0.081	0.606±0.062	0.624±0.042	**0.638**±0.040
Total Avg		0.397±0.937	0.409±0.089	**0.427**±0.076	0.608±0.058	0.620±0.047	**0.627**±0.043

In terms of AUC, the average values vary from 0.517∼0.709, 0.512∼0.711 and 0.510∼0.734 by using 10%, 20% and 50% modules which are labeled. We notice that the average performance of MASK improves when the number of available labeled modules increases. These results varies from 0.510∼0.734, which is relatively stable.

Notice that some datasets only have limited modules and the increase in the number of modules is quite small when the percentage arises from 10% to 20%. Therefore, the improvement of the performance is very small or even no when considering two different percentages. However, the increase in the number of modules is large when the percentage arises from 10% to 50% and the improvement of the performance is high. This indicates more labeled modules will result in higher performance of MASK.

In general, the more the number of available labeled modules, the higher performance of MASK. *Therefore, high-quality dataset is a key for* MASK *to achieving good performance.*

4.5 | Threats to Validity

In this subsection, we mainly discuss the potential threats to validity of our empirical studies.

Threats to the external validity of our study are that the observed experimental results may not be applicable to other software projects. To guarantee the representative of experimental subjects, we choose the three extensively used datasets AEEEM, MOPRH and ReLink[39,61,51,62,63,38]. In addition, we choose Logistic regression as the underlying classifier for MASK and other four baseline approaches. Since Logistic regression is the most used classifier in previous SDP research[22,23,37].

Threats to the internal validity are that defects may exist in the implementation of these approaches. To minimize the internal threats, we review the codes by pair programming and make full use of mature third-party libraries such as MALSAR[64].

Threats to the construct validity are that the performance evaluation may not be representative of the real-world requirements for software defect prediction. To minimize the threats, we use the F1 measure, which has been widely used in current SDP studies[65,37,39,66,67,51,62,68] and makes a good balance between the precision and recall measures. AUC measure has also been widely used in current studies for software defect prediction since it is insensitive to class imbalanced data and does not depend on an arbitrarily-selected threshold[47,38].

5 | DISCUSSION

5.1 | Some limitations of MASK

Within-project defect prediction assumes that we have sufficient labeled data from the same project, while cross-project defect prediction assumes that we have plenty of labeled data from source projects. However, in practice, we might only have limited labeled data from both the source and target projects in some scenarios. Therefore, in this paper, we propose a new scenario,

in which both the source project and the target project have limited labeled data. Then we proposed a novel multi-task defect prediction approach MASK based on multi-task learning. To verify the effectiveness of MASK, we design and conduct experimental studies on 18 widely used real-world projects. In our experiments, all the projects to be predicted are treated as the target projects. Besides, to simulate that each target project only has a few labeled data, we propose a new experimental setting (i.e., inverse 10-fold cross validation), which means that only 10% of modules are labeled and the remaining 90% of modules are not labeled.

Based on the finally experimental results, we find several limitations for MASK. First, MASK can perform significantly better than the state-of-the-art baselines but the improvements over baselines are not very large. One of the potential reasons is the limited labeled data for some projects. For example, Apache has 19 labeled modules, Safe has 6 labeled modules and ZXing has 40 labeled modules respectively when we use 10% labeled modules to build multiple prediction models. However, we think this new scenario is important and can find applications in real software development. In the future, we encourage more researchers to propose more novel solutions (i.e., solutions based on MTL) to further improve the performance. Second, MASK can outperform STL by utilizing the relatedness between all projects. However, when applying multi-task learning to this scenario, it is a challenge to measure the relatedness for different projects. In this paper, we simply consider two heuristics for measuring the relatedness. One is whether these projects are developed by the same organization. The other is whether these projects consider the same metrics (i.e. features) to measure the extracted program modules. Finding more effective methods to measure the relatedness for different projects can help to improve the performance of our proposed MASK method. This is an open problem and we encourage more researchers to investigate this issue in the future.

5.2 | Computational Cost Analysis

MASK consists of a differential evolution optimization phase and a multi-task learning phase. The former phase aims to find optimal weights for shared and non-shared information in related projects, while the latter phase builds prediction models for each project simultaneously. The former phase is time-consuming and we want to analyze the computational cost of our method MASK and make a comparison between MASK and all baseline methods. We collect the sum of running time of $10 \times$ inverse 10-fold cross validation for each baseline method and MASK on three datasets. Notice that the gathered running time includes data preprocessing, model construction on the training set and the model application on the testing set. All the methods experimented in this paper are executed on CentOS Linux 7 (Memory: 32 GB, Processor: Intel Xeon(R) CPU E5-2609 v2 @ 2.50GHz $\times$ 8). The final results can be found in Table 9.

In Table 9, the first column lists the project name, and the second column lists the project names. The following five columns represent five methods investigated in this paper respectively. Notice that the running time in each cell represents the total time

TABLE 9 Computational Cost of Different Methods on AEEEM , MOPRH and ReLink (Unit: Second).

Datasets	Related Projects	MASK	STL	SCL	Peters Filter	Burak Filter
AEEEM	EQ	709.87	1.16	5.32	183.92	230.42
	JDT		2.86	6.04	134.83	265.64
	LC		2.36	3.79	153.42	276.03
	ML		3.78	12.41	164.03	440.74
	PDE		3.49	9.20	207.10	521.41
Sum		709.87	13.64	36.77	843.30	1734.24
MORPH	ant-1.3	556.63	1.21	1.24	58.52	80.25
	arc		1.34	1.31	60.09	119.16
	camel-1.0		1.47	1.40	61.46	119.21
	poi-1.5		1.49	1.73	69.06	129.58
	redaktor		1.34	1.05	55.99	92.80
	skarbonka		1.19	0.56	47.41	51.48
	tomcat		2.22	4.56	82.31	229.81
	velocity		1.37	1.17	63.95	95.69
	xalan-2.4		1.62	4.32	76.91	207.80
	xerces-1.2		1.82	2.51	76.45	157.39
Sum		556.63	15.08	19.84	652.15	1283.17
Relink	Apache	269.88	1.09	0.30	10.82	18.45
	Safe		0.92	0.13	10.63	12.36
	Zxing		1.14	0.62	9.86	28.35
Sum		269.88	3.16	1.05	31.31	59.16
Total		1536.37	31.88	57.66	1526.76	3076.57

of 10 $\times$ inverse 10-fold cross validation. For Mask, we report the total running time for each dataset since all prediction models are built simultaneously. Besides, we also report the total running time of each method on each dataset.

From Table 9, we can find that SCL and STL need the least time to build a prediction model and make corresponding performance evaluation. However, MASK, Peters Filter, and Burak Filter need much more time than STL and SCL. In particular, Burak Filter needs the most time to build a prediction model and makes performance evaluation in most cases. Therefore, the computational cost of MASK is acceptable.

5.3 | The Influence of Tuning Hyperparameters

According to the results in Section 4.4, by comparing the performance value with STL and SCL, we can find that MASK can outperform the two baseline methods significantly, which indicates, to some extent, MASK can learn the relatedness among related projects successfully and then improve the prediction performance. To further analyze the improvement of performance contributed by the tuned configuration in each training model, we conduct additional experiments to compare the performance before and after we tune the hyperparameters in MASK. The comparison results on AEEEM can be found in Table 10.

In Table 10, the first column lists the dataset name and the second column lists the project names. The following six columns represent the performances of MASK tuned by DEO and MASK configured randomly in terms of F1 and AUC, respectively. Since there exists randomness in MASK and Random, we run these methods 10 times and report the average value of these 10 runs. In the last row, we report the average performance and the percent of performance improvement. According to the results shown in Table 10, we can find that MASK tuned by DEO can obtain better performance and improve MASK configured randomly by 47% and 15% in terms of F1 and AUC on average respectively. In addition, we draw the similar conclusions on ReLink and MORPH. Therefore, the tuned configuration in MASK can improve the performance of MASK.

TABLE 10 Comparison Between the Performance of MASK Tuned by DEO And The Performance of MASK Configured Randomly on AEEEM

Datasets	Projects	F1			AUC		
		MASK	**Random**	**Improment(%)**	**MASK**	**Random**	**Improment(%)**
AEEEM	EQ	0.574	0.501	15%	0.677	0.584	16%
	JDT	0.539	0.318	69%	0.709	0.628	13%
	LC	0.365	0.154	137%	0.657	0.537	22%
	ML	0.255	0.214	19%	0.580	0.529	10%
	PDE	0.284	0.189	50%	0.592	0.513	15%
Average		0.403	0.275	47%	0.643	0.558	15%

6 | RELATED WORK

Since the target software projects usually lack the labeled modules, a possible solution is to use other historical projects with labeled modules to train the prediction models. This issue is called the cross-project defect prediction (CPDP) [39,69,65,70,11,71,72,38,73,74,75]. However, the dataset distribution of the target and source projects is usually different, which makes CPDP a challenging task. Zimmermann et al. [27] conducted a large-scale empirical study to investigate the feasibility of CPDP and their results were not optimistic.

Therefore, researchers proposed new methods to improve the performance of CPDP. Turhan et al. [16] proposed Burak filter which selects k (i.e., 10) nearest modules in the source projects for each module in the target project. Peters et al. [37] improved the filter mechanism, which took in the infra-structure of source projects. Ma et al. [65] proposed a method which assigns higher weights to the source modules that are similar to the target modules. Wang et al. [68] leveraged a representation-learning algorithm (i.e., deep learning) to learn semantic representation of the modules from the projects. Chen et al. [40] proposed a novel algorithm to remove negative modules from the source projects. Xia et al. [22] proposed a two-layer framework Hydra, which combined

the genetic algorithm and ensemble learning to capture general properties between the source and target projects and merits of multiple prediction models.

Some researchers investigate heterogeneous CPDP, which assumes that the source and target projects have different feature sets. Nam and Kim[61] proposed the heterogeneous CPDP method, including feature selection phase and feature mapping phase. Jing et al.[51] solved the problem by defining unified feature space and applying CCA (Canonical Correlation Analysis)-based transfer learning.

Other researchers considered unsupervised learning methods. Nam and Kim[62] performed defect prediction on unlabeled data using a cluster based method which has two phases. They further used feature selection and module selection to remove noises in dataset to improve CLA and proposed CLAMI. Zhang et al.[63] designed a connectivity-based unsupervised prediction model. Rectently, Zhou et al.[38] proposed two unsupervised method (ManualDown and ManualUp) and they suggested that these two simple methods should be set as baseline methods in the future CPDP research.

More studies can be found in a recent systematic literature review and meta-analysis on CPDP[11]. Different from previous studies, we firstly propose a new usage scenario, which both source and target projects have limited labeled data. Then we propose a MASK approach based on multi-task learning. Empirical results on 23 real-world software projects show the effectiveness of our proposed approach.

7 | CONCLUSION AND FUTURE WORK

In this paper, we propose a new usage scenario and then proposed a novel approach MASK, which mainly based on multi-task learning. MASK has two phases: differential evolution optimization phase and multi-task learning phase. The former phase is used to search for the optimal value of regularization parameters for both the shared information and non-shared information. The latter phase uses the optimal value of regularization parameters to build models simultaneously and outputs models for each project. To show the effectiveness of MASK, we conduct empirical studies on 18 real-world projects. The experiment results show that MASK can outperform state-of-the-art baseline approaches in most cases.

Our initial solution based on multi-task learning is promising although not perfect. Therefore, our study can inspire more researchers to propose advanced solutions for this usage scenario. In the future, we plan to evaluate MASK with datasets from more commercial and open-source projects. Moreover, we also want to improve the performance of MASK by designing more effective ways to measure the relatedness of different projects.

ACKNOWLEDGMENT

This work was supported in part by National Natural Science Foundation of China (Grant Nos.61373012, 61202006, 91218302, 61321491), The Open Project of State Key Laboratory for Novel Software Technology at Nanjing University (Grant Nos.KFKT2016B18), Jiangsu Government Scholarship for Overseas Studies, and China Scholarship Council (Grant No. 201806190172), which supports overseas studies for Chao Ni to Monash University in Australia.

References

1. Kamei Y, Shihab E. Defect Prediction: Accomplishments and Future Challenges. In: Proceedings of the IEEE International Conference on Software Analysis, Evolution, and Reengineering; 2016: 33-45.

2. Hall T, Beecham S, Bowes D, Gray D, Counsell S. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering* 2012; 38(6): 1276-1304.

3. Ghotra B, Mcintosh S, Hassan AE. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In: Proceedings of the IEEE International Conference on Software Engineering; 2015: 789-800.

4. Jing XY, Ying S, Zhang ZW, Wu SS, Liu J. Dictionary learning based software defect prediction. In: Proceedings of the 36th International Conference on Software Engineering; 2014: 414-423.

5. Li J, He P, Zhu J, Lyu MR. Software Defect Prediction via Convolutional Neural Network. In: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security; 2017: 318-328.

6. Liu W, Liu S, Gu Q, Chen X, Chen D. FECS: A Cluster Based Feature Selection Method for Software Fault Prediction with Noises. In: Proceedings of the Annual International Computers, Software and Applications Conference; 2015: 276-281.

7. Liu S, Chen X, Liu W, Chen J, Gu Q, Chen D. FECAR: A Feature Selection Framework for Software Defect Prediction. In: Proceedings of the Annual International Computers, Software and Applications Conference; 2014: 426-435.

8. Liu W, Liu S, Gu Q, Chen J, Chen X, Chen D. Empirical Studies of a Two-Stage Data Preprocessing Approach for Software Fault Prediction. *IEEE Transactions on Reliability* 2016; 65(1): 38-53.

9. Chen X, Zhang D, Zhao Y, Cui Z, Ni C. Software defect number prediction: Unsupervised vs supervised methods. *Information and Software Technology* 2019; 106: 161-181.

10. Chen X, Zhao Y, Wang Q, Yuan Z. MULTI: Multi-objective effort-aware just-in-time software defect prediction. *Information and Software Technology* 2018; 93: 1-13.

11. Hosseini S, Turhan B, Gunarathna D. A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction. *IEEE Transactions on Software Engineering* 2017; PP(99): 1-1.

12. McIntosh S, Kamei Y. Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. *IEEE Transactions on Software Engineering* 2018; 44(5): 412-428.

13. Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering* 2018; 44(6): 534-550.

14. Poon WN, Bennin KE, Huang J, Phannachitta P, Keung JW. Cross-project defect prediction using a credibility theory based naive bayes classifier. In: Proceedings of the IEEE International Conference on Software Quality. ; 2017: 434-441.

15. Yan M, Fang Y, Lo D, Xia X, Zhang X. File-level defect prediction: Unsupervised vs. supervised models. In: Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement; 2017: 344-353.

16. Turhan B, Menzies T, Bener AB, Di Stefano J. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* 2009; 14(5): 540-578.

17. Kim S, Whitehead Jr EJ, Zhang Y. Classifying software changes: Clean or buggy?. *IEEE Transactions on Software Engineering* 2008; 34(2): 181-196.

18. Menzies T, Greenwald J, Frank A. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering* 2007; 33(1): 2-13.

19. Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y. Implications of ceiling effects in defect predictors. In: Proceedings of the International Workshop on Predictor Models in Software Engineering; 2008: 47-54.

20. Hassan AE. Predicting faults using the complexity of code changes. In: Proceedings of International Conference on Software Engineering; 2009: 78-88.

21. Li Z, Jing XY, Zhu X. Heterogeneous fault prediction with cost-sensitive domain adaptation. *Software Testing, Verification and Reliability* 2018; 28(2): e1658.

22. Xia X, Lo D, Pan SJ, Nagappan N, Wang X. HYDRA: Massively Compositional Model for Cross-Project Defect Prediction. *IEEE Transactions on Software Engineering* 2016; 42(10): 977-998.

23. Ni C, Liu WS, Chen X, Gu Q, Chen DX, Huang QG. A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction. *Journal of Computer Science and Technology* 2017; 32(6): 1090-1107.

24. Ni C, Liu W, Gu Q, Chen X, Chen D. FeSCH: A Feature Selection Method using Clusters of Hybrid-data for Cross-Project Defect Prediction. In: Proceedings of the Annual International Computers, Software and Applications Conference; 2017: 51-56.

25. Krishna R, Menzies T, Fu W. Too much automation? the bellwether effect and its implications for transfer learning. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering; 2016: 122-131.

26. Li Z, Jing XY, Zhu X, Zhang H. Heterogeneous Defect Prediction Through Multiple Kernel Learning and Ensemble Learning. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution; 2017: 91-102.

27. Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering; 2009: 91-100.

28. Xu Z, Liu J, Luo X, Zhang T. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In: Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering; 2018: 209-220.

29. Caruana R. Multitask learning. *Machine learning* 1997; 28(1): 41-75.

30. Zhang Y, Yang Q. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* 2017.

31. Jalali A, Ravikumar PD, Sanghavi S, Chao R. A Dirty Model for Multi-task Learning. In: Proceedings of Neural Information Processing Systems Conference; 2010: 964-972.

32. Maurer A, Pontil M, Romeraparedes B. Sparse coding for multitask and transfer learning. In: Proceedings of the International Conference on Machine Learning; 2013: 343-351.

33. Hernández-Lobato D, Hernández-Lobato JM, Ghahramani Z. A probabilistic model for dirty multi-task feature selection. In: Proceedings of International Conference on Machine Learning; 2015: 1073-1082.

34. Han L, Zhang Y. Multi-Stage Multi-Task Learning with Reduced Rank. In: Proceedings of the AAAI Conference on Artificial Intelligence; 2016: 1638-1644.

35. Zhang Y, Yang Q. Learning Sparse Task Relations in Multi-Task Learning.. In: Proceedings of the AAAI Conference on Artificial Intelligence; 2017: 2914-2920.

36. Price K, Price K. *Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. Kluwer Academic Publishers . 1997.

37. Peters F, Menzies T, Marcus A. Better cross company defect prediction. In: Proceedings of IEEE Working Conference on Mining Software Repositories; 2013: 409-418.

38. Zhou Y, Yang Y, Lu H, et al. How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction. *ACM Transactions on Software Engineering and Methodology* 2018; 27(1): 1.

39. Nam J, Pan SJ, Kim S. Transfer defect learning. In: Proceedings of the International Conference on Software Engineering; 2013: 382-391.

40. Chen L, Fang B, Shang Z, Tang Y. Negative samples reduction in cross-company software defects prediction. *Information and Software Technology* 2015; 62: 67-77.

41. Panichella A, Oliveto R, De Lucia A. Cross-project defect prediction models: L'union fait la force. In: Proceedings of The European Conference on Software Maintenance and Reengineering and The Working Conference on Reverse Engineering; 2014: 164-173.

42. Pan SJ, Yang Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge & Data Engineering* 2010; 22(10): 1345-1359.

43. Tantithamthavorn C, Mcintosh S, Hassan AE, Matsumoto K. Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the International Conference on Software Engineering; 2016: 321-332.

44. Fu W, Menzies T. Easy over hard: a case study on deep learning. In: Proceedings of The Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering; 2017: 49-60.

45. Fu W, Menzies T, Shen X. Tuning for software analytics: Is it really necessary?. *Information & Software Technology* 2016; 76: 135-146.

46. Agrawal A, Fu W, Menzies T. What is wrong with topic modeling? And how to fix it using search-based software engineering. *Information and Software Technology* 2018; 98: 74-88.

47. Agrawal A, Menzies T. Is "Better Data" Better Than "Better Data Miners"?. In: Proceedings of the International Conference on Software Engineering; 2018: 1050-1061.

48. Vesterstrom J, Thomsen R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: Proceedings of the Congress on Evolutionary Computation; 2004: 1980-1987.

49. Torczon V. On the convergence of pattern search algorithms. *SIAM Journal on optimization* 1997; 7(1): 1-25.

50. Xia X, Feng Y, Lo D, Chen Z, Wang X. Towards more accurate multi-label software behavior learning. In: IEEE. Proceedings of Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering; 2014: 134–143.

51. Jing X, Wu F, Dong X, Qi F, Xu B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In: Proceedings of The Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering; 2015: 496-507.

52. Wu R, Zhang H, Kim S, Cheung SC. Relink: recovering links between bugs and changes. In: Proceedings of The Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering; 2011: 15-25.

53. D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches. In: Proceedings of the IEEE Working Conference on Mining Software Repositories; 2010: 31-41.

54. Yan M, Xia X, Shihab E, Lo D, Yin J, Yang X. Automating change-level self-admitted technical debt determination. *IEEE Transactions on Software Engineering* 2018.

55. Fan Y, Xia X, Lo D, Li S. Early prediction of merged code changes to prioritize reviewing tasks. *Empirical Software Engineering* 2018; 23(6): 3346-3393.

56. Fan Y, Xia X, Lo D, Hassan AE. Chaff from the wheat: Characterizing and determining valid bug reports. *IEEE Transactions on Software Engineering* 2018.

57. Yan M, Xia X, Lo D, Hassan AE, Li S. Characterizing and identifying reverted commits. *Empirical Software Engineering* 2019: 1–38.

58. Cliff N. *Ordinal methods for behavioral data analysis*. Lawrence Erlbaum Associates . 1996.

59. Wilcoxon F. Individual comparisons by ranking methods. *Biometrics bulletin* 1945; 1(6): 80-83.

60. Abdi H. Bonferroni and Šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics* 2007; 3: 103-107.

61. Nam J, Kim S. Heterogeneous defect prediction. In: Proceedings of The Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering; 2015: 508-519.

62. Nam J, Kim S. Clami: Defect prediction on unlabeled datasets. In: Proceedings of the International Conference on Automated Software Engineering; 2015: 452-463.

63. Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the International Conference on Software Engineering; 2016: 309-320.

64. J. Zhou JC, Ye J. *MALSAR: Multi-tAsk Learning via StructurAl Regularization*. Arizona State University; 2012.

65. Ma Y, Luo G, Zeng X, Chen A. Transfer learning for cross-company software defect prediction. *Information and Software Technology* 2012; 54(3): 248-256.

66. Zhang F, Mockus A, Keivanloo I, Zou Y. Towards building a universal defect prediction model. In: Proceedings of the Working Conference on Mining Software Repositories; 2014: 182-191.

67. Zhang Y, Lo D, Xia X, Sun J. An Empirical Study of Classifier Combination for Cross-Project Defect Prediction. In: Proceedings of The Annual International Computers, Software and Applications Conference; 2015: 264-269.

68. Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction. In: Proceedings of the International Conference on Software Engineering; 2016: 297-308.

69. Fukushima T, Kamei Y, Mcintosh S, Yamashita K, Ubayashi N. An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the Working Conference on Mining Software Repositories; 2014: 172-181.

70. Rahman F, Posnett D, Devanbu P. Recalling the imprecision of cross-project defect prediction. In: Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering; 2012: 1-11.

71. Turhan B, Mısırlı AT, Bener A. Empirical evaluation of the effects of mixed project data on learning defect predictors. *Information and Software Technology* 2013; 55(6): 1101-1118.

72. Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches. In: Proceedings of the 40th International Conference on Software Engineering; 2018: 1063-1063.

73. Liu C, Yang D, Xia X, Yan M, Zhang X. A two-phase transfer learning model for cross-project defect prediction. *Information and Software Technology* 2019; 107: 125-136.

74. Liu C, Yang D, Xia X, Yan M, Zhang X. Cross-Project Change-Proneness Prediction. In: Proceedings of International Computer Software and Applications Conference; 2018: 64-73.

75. Hosseini S, Turhan B, Mäntylä M. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. *Information and Software Technology* 2018; 95: 296-312.