Multi-stage Dynamic Selection for Cross-Project Defect Prediction

Juscimara G. Avelino†, Juscelino S. A. Junior†, George D. C. Cavalcanti†, Rafael M. O. Cruz‡
†Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil
‡École de Technologie Supérieure, University of Quebec, Montreal, Canada
{jga2, jsaj, gdcc}@cin.ufpe.br, rafael.menelau-cruz@etsmtl.ca

***Abstract*—Cross-Project Defect Prediction (CPDP) involves building models using data from external projects, called training projects, to predict modules from the target project. However, traditional CPDP methods suffer from the distribution shift between training and target projects that affects the model's performance. This paper proposes a novel CPDP framework that addresses this issue by proposing a two-stage multiple classifier system (MCS) selection scheme: one working at the project level and another at the module level. In the first stage, the framework evaluates multiple possible MCS configurations to find one that covers and generalizes well across multiple training projects. Consequently, the proposal is likely to obtain a diverse set of classifiers, each specialized in tackling software modules with distinct characteristics. The second selection stage operates at test time, selecting the most competent classifiers to predict each new module in the target project. Unlike previous approaches that apply the same classifiers to the entire target project, the proposed framework performs module-level model selection. This way, the system is more robust to changes in distributions between training and target projects because the selected set of classifiers is module-dependent. Our experimental results using 82 projects from four different CPDP benchmark datasets demonstrate that the proposed approach outperforms the state-of-the-art CPDP methods in most scenarios. The code, dataset, and further details about the proposed method are publicly available at https://github.com/jsaj/Multi_DES.**

I. Introduction

Software reliability is a key factor for ensuring the quality of software products. Software defects are programming problems that may arise from issues in the source code or requirements, negatively affecting software quality and reliability [1]. Software Defect Prediction (SDP) aims to support resource allocation by identifying defect-prone software modules in advance [2]. To this end, the research community has proposed several machine learning–based approaches that leverage data stored in software repositories to predict defect proneness or estimate the likelihood of future failures [3].

Although a wide range of defect prediction methods has been proposed, building accurate models requires sufficient historical defect data [4]. In practice, however, many software projects lack such data, especially newly created projects or those that do not maintain detailed defect histories [5]. As a result, the limited availability of training data remains a major challenge in software defect prediction.

To address this issue, Cross-Project Defect Prediction (CPDP) has been proposed as an alternative, in which models are trained on data from external projects (source projects) and applied to a target project [6], [7]. CPDP is particularly valuable when the target project has insufficient historical data and has therefore become an important subtopic in defect prediction research [8]. Despite its potential, CPDP methods face major challenges due to distribution shifts between source and target projects and the limited availability of supervised or unsupervised data from the target project. As such, empirical studies have shown that many state-of-the-art CPDP approaches are not significantly better than trivial predictors [8].

Several strategies have been proposed to mitigate distribution shifts in CPDP, including data transformation, source project selection, and ensemble-based weighting schemes [9]–[11]. However, distribution shifts can still significantly degrade generalization performance [12], especially in CPDP scenarios where projects differ in coding standards, development practices, and data collection procedures [1]. Most existing approaches tackle this problem globally by modeling entire project distributions and often rely on target project information to estimate shifts or learn transformations. We argue that this global perspective is limited, as CPDP is better viewed as a dynamic problem composed of multiple local regions with distinct characteristics, motivating adaptive techniques that adjust their behavior at prediction time.

In this work, we propose Multi-stage Dynamic Ensemble Selection for Cross-Project Defect Prediction (Multi-DES), a CPDP framework based on a two-stage selection scheme designed to improve robustness under distribution shifts.

Multi-DES operates at two levels. At the project level, it selects the dynamic selection technique, base classifier, and pool size that best generalize across a set of training projects using an Aggregate Rank Minimization (ARM) scheme, a multi-criteria ranking strategy that considers multiple performance metrics [8]. At the instance level, the selected configuration dynamically identifies the most competent classifiers for each software module in the target project. By relying exclusively on training projects and performing instance-based selection at inference time, Multi-DES can be applied to entirely new projects with very limited data, including scenarios with only a single software module.

The second selection stage relies on Dynamic Selection (DS) techniques [13]. DS methods assume that different classifiers act as experts in different regions of the feature space and dynamically select the most competent classifier

for each instance. Their ability to adapt the system topology at inference time makes them a promising alternative for handling distribution shifts in heterogeneous scenarios [14], [15].

Our experimental results show that the proposed Aggregate Rank Minimization (ARM) selection mechanism leads to better generalization than single-metric selection strategies. Multi-DES achieves superior or equivalent performance compared to state-of-the-art CPDP methods in most scenarios while relying exclusively on source project data.

The main contributions of this work are summarized as follows:

- A multi-stage dynamic selection framework for CPDP, named Multi-DES, that does not rely on any information from the target project distribution.
- An Aggregate Rank Minimization (ARM)–based configuration selection strategy that improves generalization compared to traditional single-metric approaches.
- A comprehensive empirical evaluation on four CPDP benchmark datasets comprising 82 software projects.

II. Related work

This section reviews the main approaches proposed for Cross-Project Defect Prediction (CPDP). First, we categorize the CPDP literature into representative groups based on their methodological characteristics. Then, we discuss dynamic classifier and ensemble selection techniques and motivate their applicability to CPDP scenarios.

CPDP Groups. Amasaki, Aman, and Yokogawa [16] categorized CPDP approaches implemented in the CrossPare library [17] into four groups: Data Transformation, Instance Weighting, Ensemble Learning, and Feature Selection. Based on more recent studies, we extend this taxonomy by adding a fifth group related to Advanced Machine Learning methods.

Data transformation methods attempt to reduce distribution shifts by normalizing or scaling metrics across projects [6], [9]. More recent approaches extend this idea by learning richer representations through semantic and syntactic encoding of software artifacts, aiming to bridge cross-project gaps at the representation level [18]. Other works select source projects that are closer to the target project according to distance measures [10], [19]. However, these approaches often require access to the target project distribution and are limited when dealing with entirely new projects or projects with very few instances.

Instance weighting approaches aim to favor training instances that are more similar to the target project. Representative works include relevance filtering using nearest neighbors [20] and local modeling strategies based on clustering [21]. Although effective in some scenarios, these methods may discard useful data and suffer from high false alarm rates, especially when instance selection is performed at coarse granularity levels [22]. Effort-aware approaches further incorporate inspection cost into the weighting and prioritization process, as demonstrated by recent studies revisiting supervised and unsupervised CPDP methods under effort-aware evaluation settings [2].

Ensemble learning has also been explored in CPDP, with methods combining classifiers trained on different projects or feature subsets [11], [23]. Nevertheless, static ensembles remain limited by distribution shifts, as only a subset of classifiers is typically competent for predicting a given module, while others may introduce noise.

Feature selection methods focus on improving transferability by removing noisy or redundant metrics [24], [25]. While such approaches simplify the learning problem, they do not explicitly address local competence or instance-level variability across projects.

More recent studies have increasingly focused on advanced machine learning techniques, particularly deep learning and transfer/domain adaptation frameworks. Examples include autoencoder-based models with dynamic adversarial adaptation [26], multi-stage cross-project frameworks based on feature representation and knowledge transfer [27], and approaches combining fuzzy embeddings with deep learning models [28]. Despite their promising results, these methods generally rely on global feature representations and implicit assumptions about source–target similarity, which may limit their robustness under severe domain mismatch.

Dynamic Classifier and Ensemble Selection. Dynamic selection (DS) techniques select classifiers based on their estimated competence for each test instance and have demonstrated superior performance over static ensembles in heterogeneous and non-stationary scenarios [13]. These methods assume classifier specialization across different regions of the feature space and dynamically adapt predictions at inference time. DS techniques are typically divided into Dynamic Classifier Selection (DCS), which selects a single classifier, and Dynamic Ensemble Selection (DES), which selects a subset of classifiers per instance. While DCS approaches may be sensitive to noise and class imbalance, DES methods are generally more robust by aggregating multiple competent classifiers, including oracle-based techniques [29] and meta-learning approaches such as META-DES [14], [30].

Critical analysis. Despite the extensive literature on CPDP, Herbold et al. [8] showed that existing approaches do not employ dynamic selection mechanisms and generally rely on global modeling assumptions that require information from the target project. Since classifier competence can vary across different regions of the feature space [13], using a single global model or static ensemble may be inadequate in heterogeneous CPDP scenarios. Dynamic selection addresses this limitation by selecting models at the instance level rather than relying on a single global representation. The proposed Multi-DES framework tackles this challenge by adopting a multi-criteria configuration selection strategy that promotes better generalization across multiple source projects and applies the selected configuration to unseen target projects.

III. The Proposed Framework: Multi-DES

Figure 1 shows the proposed framework, Multi-stage Dynamic Ensemble Selection (Multi-DES) for Cross-Project Defect Prediction, which aims to select the best model, from a predefined set of options, to predict defects in a project that was not previously used to train the system. Multi-DES is a strict cross-project defect predictor, meaning that it relies exclusively on data from source projects and does not use any information from the target project during training. In particular, the framework exploits a set of source datasets $\mathcal{T} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$, where each dataset corresponds to a distinct software project composed of multiple modules, to predict defects in a new target project $\mathcal{G}$ that did not take part in the training process. Following the taxonomy adopted by Herbold et al. [31], Multi-DES falls into the category of strict cross-project defect prediction.

Multi-DES adopts a multi-stage strategy to optimize the configurations of dynamic selection algorithms using only source projects. This optimization relies on a validation set ($\mathcal{V}$), which, similarly to $\mathcal{T}$, is composed of a set of projects, and aims to mitigate distribution differences between source and target data, one of the main challenges in CPDP systems [32]. As illustrated in Figure 1, the framework comprises three main phases: I) overproduction, II) best configuration selection, and III) prediction. In the overproduction phase, multiple DS configurations are generated by varying the base learning algorithm, the DS technique, and the pool size. The best configuration is then selected in Phase II based on its generalization ability across projects. Finally, in the prediction phase, the selected configuration is applied to the target project, dynamically adapting the ensemble to each software module and producing defect predictions for project $\mathcal{G}$.

A. Overproduction

This phase takes as input the training set $\mathcal{T}$ and the validation set $\mathcal{V}$, together with a set of dynamic selection techniques $\lambda_{list} = \{\lambda_1, \ldots, \lambda_{n_1}\}$, a set of base learning algorithms $C_{list} = \{c_1, \ldots, c_{n_2}\}$, and a set of pool sizes $M = \{m_1, \ldots, m_{n_3}\}$. For each classifier $c \in C_{list}$ and pool size $m \in M$, a classifier pool $P(c, m)$ is generated from $\mathcal{T}$ using Bagging, which promotes diversity through bootstrap sampling. Such diversity is crucial in CPDP scenarios to mitigate cross-project data heterogeneity [20].

Each pool is then evaluated on $\mathcal{V}$ using every dynamic selection strategy $\lambda \in \lambda_{list}$. For each instance $v \in \mathcal{V}$, a region of competence $\mathrm{RoC}_\lambda(g)$ is defined, and a subset of competent classifiers $P'_\lambda(g) \subseteq P(c, m)$ is selected and combined according to λ to produce a prediction l_g. The predictions obtained for all instances in $\mathcal{V}$ are then used to assess the overall performance of the corresponding dynamic selection configuration, using performance metrics such as F1-score and AUC.

This process evaluates all configurations in the Cartesian product $\lambda_{list} \times C_{list} \times M$, resulting in tuples $(\lambda, c, m, \textit{metrics}, P)$. Exploring this configuration space is crucial for CPDP, as it increases system diversity by varying competence definitions, selection criteria, and pool sizes—factors that are often fixed in prior work [13]. By systematically assessing a wide range of configurations across multiple source projects, Multi-DES identifies configurations that are more likely to generalize under cross-project distribution shifts. Importantly, this phase relies exclusively on source projects, enabling configuration selection without access to the target project $\mathcal{G}$ and allowing defect prediction for entirely new projects.

B. Best configuration selection

This phase (Algorithm 1) selects the most competent model based on the results of the experimental configurations generated in the overproduction phase. Each configuration is represented by a tuple $(\lambda, c, m, \textit{metrics}, P)$, where λ denotes the dynamic selection algorithm, P is the pool of classifiers trained using the base learning algorithm c with pool size m, and *metrics* stores the evaluation results, e.g., F1-score, AUC, and False Alarm.

Algorithm 1 Best configuration selection using Aggregate Rank Minimization (ARM)

Require: results: list of tuples $(\lambda, c, m, \textit{metrics}, P)$
1: $rankings = \varnothing$
2: **for each** *metric* $\in$ *metrics* **do**
3: $\quad values =$ values-per-metric(*metrics.metric*)
4: $\quad rankings[metric] =$ *rankdata*$(values)$
5: **end for**
6: *avg_ranking* $=$ compute-average-rank($rankings$)
7: *low_rank* $=$ select-lower-average(*avg_ranking*)
8: $\lambda^*, P^* =$ select-config(results, *low_rank*)
9: **return** λ^*, P^*

Different performance metrics are considered in this phase due to the importance of evaluating models from multiple perspectives, as approaches may exhibit varying behavior depending on the criterion adopted [8]. Accordingly, we propose an *Aggregate Rank Minimization (ARM)* strategy for configuration selection. ARM operates as follows: for each performance metric, configurations are ranked according to their performance, with the best-performing configuration receiving rank 1, the second-best rank 2, and so forth (lines 2–5). In the presence of ties, a mid-rank (fractional ranking) strategy is adopted, assigning each tied configuration the average of the ranks they would occupy.

After computing rankings for all metrics, ARM calculates the average rank of each configuration across metrics (line 6). The configuration with the lowest average rank (line 7) is selected, as it represents the configuration with the best overall trade-off among the considered metrics and, consequently, the highest expected generalization ability (line 8).

By aggregating rankings across multiple evaluation criteria, ARM balances competing performance objectives and reduces the risk of overfitting to a single metric. As a result, it increases the likelihood of selecting configurations that perform robustly when transferred to unseen target projects.

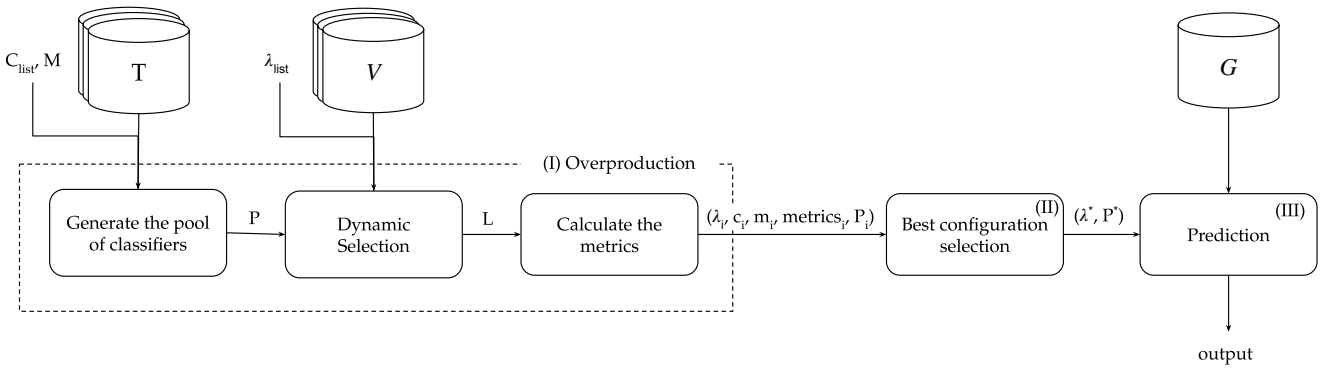

Fig. 1: Proposed Multi-DES architecture and its three phases: I) Overproduction, II) Best configuration selection, and III) Prediction. $\mathcal{T}$, $\mathcal{V}$, and $\mathcal{G}$ are the training, validation, and test datasets, respectively. C_{list} is a list of learning algorithms, M is the set with different pool sizes used to generate the pool P, λ_{list} is a list of dynamic selection techniques, P is the pool of classifiers, and L stores the output of the dynamic selection module.

C. Prediction

The *Best configuration selection* phase returns an optimal configuration defined as a pair (λ^*, P^*), where λ^* denotes the selected dynamic selection strategy and P^* the corresponding pool of classifiers. This configuration is then applied to predict defects in the target project $\mathcal{G}$.

Formally, for each instance $g \in \mathcal{G}$, a region of competence $\mathrm{RoC}(g)$ is computed based on the validation set $\mathcal{V}$. The region of competence is defined as a subset of instances in $\mathcal{V}$ that are considered relevant for assessing the competence of classifiers with respect to g, according to the dynamic selection strategy in use. Given $\mathrm{RoC}(g)$, a subset of classifiers $P'(g) \subseteq P^*$ is selected such that each classifier in $P'(g)$ demonstrates adequate competence over $\mathrm{RoC}(g)$. The dynamic selection strategy λ^* is then applied to select or combine classifiers from $P'(g)$ in order to produce the predicted label $r_g \in \{0, 1\}$, indicating the absence or presence of a defect in instance g.

IV. Experimental Protocol

Datasets. Experiments were conducted on four widely used public defect prediction datasets: (1) PROMISE [33], comprising 62 product versions from 31 projects; (2) RELINK [34], containing defect data from three projects; (3) NASA (MDP), a pre-processed version of the Metrics Data Program with 12 projects [35]; and (4) AEEEM [36], which includes five Java products from different projects. Each dataset provides a distinct set of software metrics used as features. To handle scale differences, Z-score normalization is applied using statistics computed from the training set and then reused to normalize test instances.

A leave-one-project-out evaluation protocol is adopted, where one project is used as the target and the remaining projects for training. When a project is selected as the target, all its versions are excluded from the training set. The process is repeated for all projects, and the average and standard deviation of the results are reported.

The approaches are evaluated using three performance metrics—F1-score, Area Under the ROC Curve (AUC), and False Alarm—and statistical significance is assessed using the Wilcoxon Signed-Rank test.

Classifiers and Dynamic Selection Techniques. Dynamic selection techniques rely on a pool of base classifiers; thus, we employed *Decision Tree* (DT), *Logistic Regression* (LR), *Naive Bayes* (NB), and *Random Forest* (RF), which are widely used in CPDP and dynamic ensemble learning [8], [13]. All classifiers were implemented using scikit-learn[1] with default hyperparameters. The classifier pool was generated using Bagging, the standard approach in the dynamic selection literature [13], [30], with pool sizes ranging from 10 to 100 in steps of 10.

Eight state-of-the-art dynamic selection techniques were considered: *K-Nearest Oracles Union* (KNU) and *K-Nearest Oracles Eliminate* (KNE) [29], *k-Nearest Output Profiles* (KNOP) [37], *META-DES* [30], *Overall Local Accuracy* (OLA) and *Local Class Accuracy* (LCA) [38], *Multiple Classifier Behavior* (MCB) [39], and *Modified Classifier Rank* (Rank) [40]. These techniques were selected based on citation impact, diversity of selection criteria (oracle, behavior, accuracy, and meta-learning), reported performance, and availability in DESlib[2] [41].

State-of-the-art CPDP approaches. The proposed framework is compared with four representative CPDP methods—*CamargoCruz09-DT* (CC09-DT) [9], *Turhan09-DT* (T09-DT) [20], *Menzies11-RF* (M11-RF) [21], and *Watanabe08-DT* (W08-DT) [6]—which were identified as top-performing techniques in the large-scale comparative study by Herbold et al. [8]. The experimental results of these methods were obtained from the publicly available replication package[3] . In addition, we compare our framework with the Effort-Aware Supervised Cross-project method [2], using Naive Bayes as the base classifier (EASC-NB), which was proposed after the study by Herbold et al [8].

[1]https://scikit-learn.org/ — version 1.0.2
[2]https://github.com/scikit-learn-contrib/DESlib — version 0.3.5
[3]https://github.com/sherbold/replication-kit-tse-2017-benchmark

V. Experiments

The experimental analysis evaluates the proposed Multi-DES framework from three complementary perspectives. First, it examines whether the multi-stage design effectively exploits the diversity generated in the overproduction phase by analyzing the contribution of different configurations across projects. Second, it assesses the effectiveness of the proposed Ranking strategy for configuration selection, focusing on its ability to identify models with higher generalization capability through the joint consideration of multiple evaluation metrics. Finally, Multi-DES is compared with state-of-the-art CPDP approaches to analyze its relative performance across datasets and evaluation metrics.

A. Configuration analysis

In the Overproduction phase, four base learning algorithms (C_{list}), eight dynamic selection techniques (λ_{list}), and then pool sizes (M) are evaluated, generating a total of 320 ($4 \times 8 \times 10$) configurations. After that, only one configuration is selected in the second phase, "Best configuration selection", per project under evaluation. These configurations compose diverse classification systems that create different decision spaces.

Figure 2 shows the number of times each configuration is selected. Since the leave-one-project-out procedure is employed, 82 configurations are selected, i.e., one per project. For the base classifiers (Figure 2(a)), Random Forest deserves a remark, being selected 69 out of 82 times. However, other base classifiers were the best option depending on the project under evaluation. Regarding the dynamic selection technique (Figure 2(b)), LCA, OLA, and META-DES reached the top three methods, being selected 29, 16, and 15 times, respectively. Concerning the pool size (Figure 2(c)), no clear preference can be observed, as the distribution is nearly uniform. This behavior is consistent with previous in-depth analyses of dynamic ensemble selection frameworks, which have shown that different pool sizes often lead to comparable performance, with observed variations largely attributable to randomness rather than systematic effects.

Important to highlight that all the possible configurations of base classifier, dynamic selection method, and pool size were selected at least once. In particular, there was no clear better option for the dynamic selection method and the pool size. These results show the relevance of having a procedure to search for the best configuration instead of adopting a static classifier, since the best configuration differs from task to task.

B. Selection approach evaluation

Four strategies were evaluated to select the best configuration in Phase II of the Multi-DES framework: three single-metric strategies optimizing F1-score, AUC, or False Alarm, and the proposed Aggregate Rank Minimization (ARM) strategy, which combines multiple metrics to favor configurations with better generalization capability.

The results show that ARM achieves superior or comparable performance to single-metric strategies across most datasets and evaluation metrics. Pairwise Wilcoxon signed-rank tests indicate that ARM outperforms or matches single-metric selection in the majority of cases ($p < 0.05$), particularly when compared with False Alarm–based selection, where ARM consistently identifies more effective configurations.

These findings reveal a key limitation of single-metric selection, as optimizing a configuration for a single criterion may lead to overfitting and reduced robustness on unseen projects, reinforcing the need for multi-perspective evaluation in CPDP [8]. To further contextualize its effectiveness, ARM is compared with an upper-bound scenario (*Truth*), which represents an idealized retrospective selection based on test performance. Although infeasible in practice, this comparison shows that ARM often closely approximates the upper bound, especially for AUC, while larger gaps in some cases (e.g., False Alarm on RELINK) indicate opportunities for improvement.

Overall, the results confirm that no single configuration is optimal across all projects and highlight the effectiveness of the proposed multi-stage selection scheme, supporting the adoption of ARM as the default selection strategy in subsequent analyses.

C. Multi-DES versus state-of-the-art

Table I presents the performance of Multi-DES and state-of-the-art CPDP methods across four datasets. The best results per dataset are highlighted in bold, and the second best are underlined.

For the F1-score, Multi-DES achieved the best performance on the AEEEM and RELINK datasets. Regarding AUC, Multi-DES obtained the best results on AEEEM and NASA, and second-best results on PROMISE and RELINK. In the latter two datasets, the differences between Multi-DES and EASC are small (around 1%), while in AEEEM and NASA, Multi-DES outperformed EASC by approximately 10%. For the False Alarm metric, Multi-DES achieved the best or second-best performance in most datasets, indicating robust behavior across different scenarios. Notably, on the AEEEM dataset, Multi-DES achieved the best results for all metrics and was never the worst-performing method.

The Wilcoxon signed-rank test results reported in the last row of each metric block in Table I indicate that Multi-DES is statistically superior in most pairwise comparisons, particularly for the AUC and False Alarm metrics.

Table II summarizes the aggregated win–tie–loss counts comparing Multi-DES against each baseline across all datasets. For AUC, Multi-DES clearly dominates the competing methods, while for False Alarm, it also achieves a large number of wins against most baselines. Although the F1-score presents a more balanced scenario, Multi-DES remains competitive and achieves comparable or superior performance to the literature methods.

Considering that the maximum number of wins per metric is 82, resulting in 246 wins across all three metrics, Multi-DES achieved more than 70% of the wins against CamargoCruz09-DT, Menzies11-RF, Turhan09-DT, and Watanabe08-DT, and a

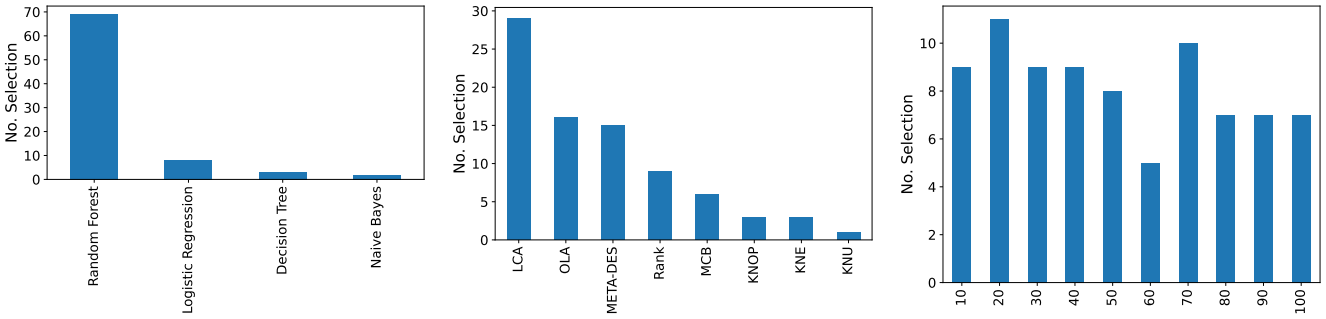

Fig. 2: Configuration selection frequency: (a) base classifier, (b) dynamic selection method, and (c) pool size.

TABLE I: Performance (mean±variance) of Multi-DES and CPDP methods across datasets. The last row of each metric reports the p-values of the Wilcoxon signed-rank test comparing Multi-DES against each baseline across all datasets. Best values per dataset are in bold and second best are underlined.

Metric	Dataset	CamargoCruz09-DT	EASC-NB	Menzies11-RF	Turhan09-DT	Watanabe08-DT	Multi-DES
F1-score ↑	AEEEM	0.312±0.007	0.372±0.010	0.269±0.008	0.271±0.004	0.301±0.000	**0.385±0.009**
	NASA	0.089±0.007	0.150±0.011	0.142±0.003	**0.161±0.007**	0.109±0.002	0.140±0.006
	PROMISE	0.367±0.023	0.270±0.020	0.328±0.016	0.359±0.029	**0.368±0.014**	0.316±0.022
	RELINK	0.543±0.003	0.242±0.118	0.555±0.009	0.528±0.057	0.489±0.022	**0.560±0.015**
	Wilcoxon p	0.2382	**0.0237**	0.7939	0.3421	0.1763	–
AUC ↑	AEEEM	0.604±0.005	0.692±0.006	0.578±0.001	0.530±0.000	0.588±0.004	**0.755±0.002**
	NASA	0.697±0.013	0.666±0.006	0.541±0.001	0.622±0.004	0.669±0.009	**0.737±0.008**
	PROMISE	0.582±0.010	**0.713±0.013**	0.574±0.005	0.588±0.012	0.593±0.008	0.709±0.012
	RELINK	0.648±0.001	**0.748±0.006**	0.653±0.001	0.635±0.006	0.602±0.011	0.734±0.010
	Wilcoxon p	**0.000***	0.5494	**0.000***	**0.000***	**0.000***	–
False Alarm ↓	AEEEM	0.063±0.001	0.174±0.021	0.047±0.002	0.131±0.009	0.109±0.002	**0.035±0.000**
	NASA	**0.013±0.000**	0.021±0.000	0.034±0.001	0.052±0.002	0.019±0.000	0.019±0.000
	PROMISE	0.203±0.012	**0.063±0.003**	0.171±0.008	0.178±0.010	0.262±0.019	0.097±0.006
	RELINK	0.212±0.006	0.170±0.058	0.211±0.002	**0.167±0.015**	0.168±0.004	0.190±0.005
	Wilcoxon p	**0.000***	0.0023	**0.000***	**0.000***	**0.000***	–

↑ indicates higher is better; ↓ indicates lower is better. Bold p-values indicate statistical significance at $\alpha = 0.05$. * indicates $p < 10^{-5}$.

TABLE II: Aggregated win–tie–loss results comparing Multi-DES against literature CPDP methods across all datasets.

Metric	CamargoCruz09-DT	EASC-NB	Menzies11-RF	Turhan09-DT	Watanabe08-DT
F1-score	40–2–40	43–4–35	41–4–37	33–2–47	37–2–43
AUC	70–1–11	42–1–39	76–0–6	77–0–5	72–0–10
False Alarm	61–6–15	22–13–47	65–9–8	69–9–4	69–5–8

comparable but slightly higher number of wins than EASC-NB.

It is worth noting that Multi-DES builds and selects models exclusively using training data, unlike several CPDP approaches that rely on information extracted from the target project for model selection or data transformation, which may lead to data leakage. Despite this restriction, Multi-DES achieves superior or equivalent results without leveraging any information from the test project. Moreover, due to its local and instance-based dynamic selection strategy, Multi-DES does not make assumptions about the target project distribution, making it particularly suitable for defect prediction in new projects with limited available data.

VI. Conclusion

Software defect prediction remains challenging due to limited labeled data, motivating the use of Cross-Project Defect Prediction (CPDP) to leverage external projects. This paper proposes Multi-DES, a multi-stage dynamic ensemble selection framework that selects model configurations using only source projects and adapts predictions at the instance level. In particular, we introduce the proposed *Aggregate Rank Minimization (ARM)* strategy for configuration selection, which improves generalization by jointly considering multiple performance metrics. The experimental results show that ARM improves generalization and that Multi-DES achieves superior or comparable performance to state-of-the-art CPDP methods across most datasets and metrics, particularly under distribution shifts, despite a single exception on the PROMISE dataset for False Alarm. The framework requires no information from the target project, enabling its use on completely new projects. Validity threats were addressed through a leave-one-project-out protocol on 82 projects, widely used evaluation metrics, and appropriate statistical tests. Future work includes reducing computational cost via configuration prediction, extending Multi-DES to online scenarios, and exploring its applicability to other defect prediction settings.

REFERENCES

[1] M. S. Rawat and S. K. Dubey, "Software defect prediction models for quality improvement: a literature study," *International Journal of Computer Science Issues*, vol. 9, no. 5, p. 288, 2012.

[2] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu, "Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 786–802, 2020.

[3] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[4] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[5] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 316–329, 2007.

[6] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *International Workshop on Predictor Models in Software Engineering*, 2008, pp. 19–24.

[7] S. Herbold, "Training data selection for cross-project defect prediction," in *International Conference on Predictive Models in Software Engineering*, 2013, pp. 1–10.

[8] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. 44, no. 9, pp. 811–833, 2017.

[9] A. E. C. Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 460–463.

[10] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, pp. 125–136, 2019.

[11] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, 2014, pp. 164–173.

[12] D. A. Cieslak and N. V. Chawla, "A framework for monitoring classifiers' performance: when and why failure occurs?" *Knowledge and Information Systems*, vol. 18, no. 1, pp. 83–108, 2009.

[13] R. M. Cruz, R. Sabourin, and G. D. Cavalcanti, "Dynamic classifier selection: Recent advances and perspectives," *Information Fusion*, vol. 41, pp. 195–216, 2018.

[14] B. Jiao, Y. Guo, D. Gong, and Q. Chen, "Dynamic ensemble selection for imbalanced data streams with concept drift," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[15] P. Zyblewski, R. Sabourin, and M. Woźniak, "Preprocessed dynamic classifier ensemble selection for highly imbalanced drifted data streams," *Information Fusion*, vol. 66, pp. 138–154, 2021.

[16] S. Amasaki, H. Aman, and T. Yokogawa, "An exploratory study on applicability of cross project defect prediction approaches to cross-company effort estimation," in *ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, 2020, pp. 71–80.

[17] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," in *International Conference on Software Engineering*, 2018, pp. 1063–1063.

[18] S. Jiang, Y. Chen, Z. He, Y. Shang, and L. Ma, "Cross-project defect prediction via semantic and syntactic encoding," *Empirical Software Engineering*, vol. 29, no. 4, p. 80, 2024.

[19] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[20] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[21] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *IEEE/ACM International Conference on Automated Software Engineering*, 2011, pp. 343–351.

[22] B. L. Sinaga, S. Ahmad, and Z. A. Abas, "A review of training data selection in software defect prediction," 2020.

[23] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *IEEE Annual Computer Software and Applications Conference*, vol. 2, 2015, pp. 264–269.

[24] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," in *Euromicro Conference on Software Engineering and Advanced applications*, 2015, pp. 96–103.

[25] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[26] W. Zhang, J. Zhao, G. Qin, and S. Wang, "Cross-project defect prediction based on autoencoder with dynamic adversarial adaptation," *Applied Intelligence*, vol. 55, no. 5, p. 324, 2025.

[27] Y. Zou and H. Wang, "A three-stage cross-project defect prediction framework based on feature representation and knowledge transfer," *Complex & Intelligent Systems*, vol. 11, no. 11, p. 459, 2025.

[28] M. Azzeh and M. J. Abdel-Rahman, "Cross-project software defects prediction using fuzzy embedding and deep learning," *Information and Software Technology*, vol. 190, p. 107968, 2026.

[29] A. H. Ko, R. Sabourin, and A. S. Britto Jr, "From dynamic classifier selection to dynamic ensemble selection," *Pattern recognition*, vol. 41, no. 5, pp. 1718–1731, 2008.

[30] R. M. Cruz, R. Sabourin, G. D. Cavalcanti, and T. I. Ren, "META-DES: A dynamic ensemble selection framework using meta-learning," *Pattern recognition*, vol. 48, no. 5, pp. 1925–1935, 2015.

[31] S. Herbold, A. Trautsch, and J. Grabowski, "Global vs. local models for cross-project defect prediction," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1866–1902, 2017.

[32] Z. Li, H. Zhang, X.-Y. Jing, J. Xie, M. Guo, and J. Ren, "Dssdpp: Data selection and sampling based domain programming predictor for cross-project defect prediction," *IEEE Transactions on Software Engineering*, pp. 1–23, 2022.

[33] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–10.

[34] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *ACM SIGSOFT Symposium and the European Conference on Foundations of Software Engineering*, 2011, pp. 15–25.

[35] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[36] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *IEEE Working Conference on Mining Software Repositories*, 2010, pp. 31–41.

[37] P. R. Cavalin, R. Sabourin, and C. Y. Suen, "Dynamic selection approaches for multiple classifier systems," *Neural Computing and Applications*, vol. 22, no. 3, pp. 673–688, 2013.

[38] K. Woods, W. P. Kegelmeyer, and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 405–410, 1997.

[39] G. Giacinto and F. Roli, "Dynamic classifier selection based on multiple classifier behaviour," *Pattern Recognition*, vol. 34, no. 9, pp. 1879–1882, 2001.

[40] M. Sabourin, A. Mitiche, D. Thomas, and G. Nagy, "Classifier combination for hand-printed digit recognition," in *International Conference on Document Analysis and Recognition*, 1993, pp. 163–166.

[41] R. M. Cruz, L. G. Hafemann, R. Sabourin, and G. D. Cavalcanti, "DESlib: A dynamic ensemble selection library in python." *Journal of Machine Learning Research*, vol. 21, no. 8, pp. 1–5, 2020.