Journal of Information Systems and Informatics
Vol. 7, No. 3, September 2025 e-ISSN: 2656-4882 p-ISSN: 2656-5935
DOI: 10.51519/journalisi.v7i3.1171 Published By DRPM-UBD

Ensemble Learning for Software Defect Prediction: Performance, Practicality and Future Directions

Bassey Isong[1], Ekoro Ekoro Igo[2]

[1]Computer Science Department, North-West University, Mafikeng, South Africa
[2] Computer Science Education Department, University of Education & Entrepreneurship, Akamkpa, Nigeria
Email: [1]bassey.isong@nwu.ac.za, [2]ekoro.ekoro@crs.coeakamkpa.edu.ng

Abstract

Ensemble learning is a leading approach in software defect prediction (SDP), offering improved predictive performance on imbalanced and high-dimensional datasets. Despite growing research interest, persistent gaps remain in model interpretability, generalizability, and reproducibility, limiting its practical adoption. This paper presents a comprehensive analysis of 56 peer-reviewed studies published between 2020 and 2025, spanning both journal and conference venues. Findings show that ensemble methods, especially when combined with sampling, feature selection, or optimisation, consistently outperform single classifiers on important metrics such as F1-score, area under the curve, and Matthew correlation coefficient. Nonetheless, few studies incorporate explainability frameworks, effort-aware evaluation, or cross-project validation. Additionally, most models are static, rely on within-project testing, and depend on legacy datasets such as PROMISE and NASA, which limit external validity. Building on this synthesis, the review highlights future research priorities, including interpretable ensemble architectures, adaptive modelling, dynamic imbalance handling, semantic feature integration, and real-time prediction. Standardised benchmarks, transparent, scalable designs are recommended to bridge the gap between experimental performance and deployment-ready SDP solutions.

Keywords: Software Defect Prediction, Ensemble Methods, Class Imbalance, Machine Learning.

1. INTRODUCTION

As software systems increase in size and complexity, ensuring product quality has become both more critical and more difficult. Modern applications often consist of interconnected components, rapid release cycles, and constantly evolving codebases, factors that strain traditional quality assurance methods. In this context, software defects remain a major concern, often leading to system failures, security risks, and higher maintenance costs [1]. Moreover, post-development testing is

often inadequate, as it tends to identify critical defects late, when fixes are most costly. Studies indicate that fixing defects in later stages of development can be up to 100 times more expensive than addressing them during the design or coding phases [1]. To minimise these risks, software defect prediction (SDP) has become a proactive and cost-effective quality assurance strategy [1][2]. The impact of SDP on a software company is shown in Fig. 1 as reported by raygun.com, an error tracking and reporting software [3]. A report released in September 2018 by the Consortium for IT Software Quality (CISQ), software failures or poor-quality software cost the U.S. approximately $2.84 trillion [2][4]. Accordingly, a significant portion of these losses stems from rework and defect correction, with up to 50% of developer time spent on fixing issues, particularly in organisations with low process maturity. As shown in Fig. 1., 41% were from reproducing bugs in the software module, 23% is the cost of correcting the errors after reproduction, 23% is the effort to design effective testcases for defect localization, 9% is the costs for accessing necessary diagnostic tools and 5% is the cost of automating the testing process [2],[3].

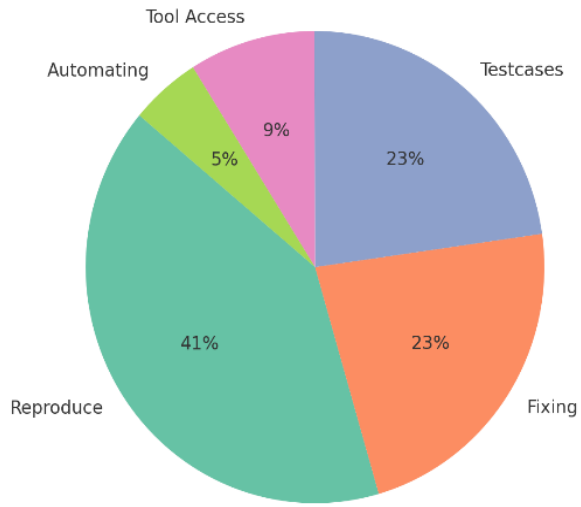

Figure 1. Cost distribution in SDP

SDP or software fault prediction (SFP) constitutes an important research area in software engineering (SE), aimed at identifying modules likely to contain defects early in the development lifecycle. Using historical code metrics, process data, and change records, these models estimate the fault-proneness of software modules [1],[4]. This approach shifts quality assurance from reactive testing to targeted prevention, helping teams focus limited resources on high-risk areas. By flagging these early in the development process, SDP helps optimise testing efforts, reduce cost and time, and improve overall reliability and user satisfaction [1][4-7]. Despite

its importance, SDP faces several challenges. Traditional defect prediction approaches relied on manual inspection and simple statistical methods, which often could not scale or adapt to large or evolving software projects. In addition, issues such as class imbalance, evolving practices, and project variability limited their effectiveness [4-12]. In response, machine learning (ML), especially ensemble methods that combine multiple models to improve accuracy and generalisation, has gained traction for building more robust and generalizable prediction models. SDP utilises ML to flag vulnerable modules early, which studies have shown can reduce manual validation effort by 30 –70% [4-12].

ML techniques have been widely adopted in SDP to address these challenges and build predictive models that categories software modules into defective and non-defective based on historical data [1],[4-12]. ML algorithms, both traditional and deep learning (DL) based, have shown effectiveness; however, relying solely on single (or base) ML models for SDP often leads to limited performance [4-12]. These models often struggle with non-linear patterns, perform poorly on imbalanced data, and can be unstable; small changes in input may lead to different results. Some tend to overfit (like deep networks), while others underfit (like linear models), limiting their overall reliability, and prompting researchers to explore more robust alternatives [10], [11]. Ensemble learning (EL) further improves SDP by combining multiple base learners or models to deal with the limitations in single-model approaches [1][4-7]. They reduce variance, correct bias, and handle class imbalance more effectively than individual models [4-7]. In SDP, datasets often contain far more non-defective than defective modules, and project variability further complicates prediction[4-7]. EL mitigates these issues by introducing diversity among learners in features, training instances, or model parameters. The main EL techniques: Bagging, boosting, and stacking, offer different trade-offs between bias and variance, making them suitable for a range of SDP scenarios [4–7]. Their effectiveness depends on balancing diversity and accuracy, a challenge particularly relevant for heterogeneous or imbalanced project datasets [1],[6]. Ensemble methods have demonstrated strong predictive performance across different domains, highlighting their potential for improved defect prediction in real-world SE settings.

Over the years, several EL-based SDPs have been designed, both homogeneous (such as bagging, boosting, or Random Forest (RF)) and heterogeneous ensembles which combine classifiers from different algorithms to increase diversity [1][4-7]. Despite growing research interest, several challenges continue to limit ensemble methods' practical application. Therefore, this paper presents a comprehensive review of ensemble-based SDP methods, focusing on their effectiveness, scalability, and practical applicability. It analyses standard datasets and benchmarks to evaluate performance, interpretability issues, and the impact of various base

learners, and identifies research gaps to guide future work in improving software quality assurance. Our main contributions include:

1) Present an overview of SDP and EL methods for SDP, including a detailed taxonomy.
2) Analyse 56 empirical studies, highlighting the ensemble type, the methodology used, including base learners, class imbalanced techniques, datasets, evaluation metrics and limitations.
3) Identify high-impact research directions to advance ensemble-based SDP.

The remaining parts of the paper are structured as follows: Section 2 presents important background and related works, Section 3 presents the study methodology, Section 4 presents the findings, their discussion, possible research directions, and study limitations. Section 5 concludes the paper.

2. BACKGROUND INFORMATION

2.1. Software Defect Prediction

SDP considers quality assurance as a supervised learning task, using labelled historical software data to identify potentially defective components. The approach assumes that metrics derived from source code, development processes, and software evolution provide sufficient information to differentiate between defect-prone and non-defect modules [1],[4],[8]. SDP itself varies in granularity, from coarse levels (e.g., classes or files) to fine-grained targets like functions or individual lines of code [1]. File-level offers broad insights for allocating testing resources while class-level strikes a balance between detail and manageable feature complexity, especially in object-oriented systems. In addition, the method level gives finer detail but struggles with scalability. Statement-level is the most precise but remains largely experimental due to the high computational cost. Also, prediction models differ temporally. Within-project models rely on historical data from the same project, capturing project-specific patterns. On the other hand, cross-project models aim to generalise across projects with limited local data. Hybrid approaches combine both to balance specificity and broader applicability.

During SDP, software metrics are used to support defect prediction models and are typically grouped into several categories based on their source and purpose. (See Table 1) Static code metrics assess structural properties like complexity, coupling, and cohesion without executing code; examples include the Chidamber-Kemerer or CK metric suite and Halstead metrics [7][8]. Likewise, process/historical metrics capture aspects of development activity, like code churn, developer experience, and change frequency, that are closely linked to the likelihood of defects. It also uses version control data to assess change patterns, bug fix frequency, and component stability [8]. Network metrics assess

architectural structure through code dependency graphs, focusing on modularity and interaction complexity. More recently, semantic metrics have been introduced, using natural language processing (NLP) on texts such as comments and documentation to improve feature richness.

Table 1. Software metrics for defect prediction

Metric	Code	Process/history	Architectural graphs	Texts
Static code	✓	✗	✗	✗
Change/process	✗	✓	✗	✗
Network	✗	✗	✓	✗
Semantic/NLP	✗	✗	✗	✓

2.2. Machine Learning and Ensemble Learning for SDP

Over the years, ML-based SDP has evolved from basic statistical models to advanced ensemble techniques. Common single-model approaches for automated and data-driven defect prediction include the traditional ML algorithms such as K-Nearest Neighbours (KNN), Naïve Bayes (NB), Decision Trees (DT), Support Vector Machines (SVM), Logistic Regression (LR), and RF, as well as DL-based approaches like Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) [1],[4-7]. Early methods like LR and NB were limited by strict data assumptions, as well as NB requiring large datasets and suffering from black-box interpretability issues. DT became popular for their interpretability and the ability to handle various feature types, but it is prone to overfitting. Similarly, SVM offered better performance on complex data using non-linear boundaries but required careful parameter tuning [1]. Neural networks have shown potential but were limited by overfitting and insufficient training data. Consequently, ensemble methods later mitigated these issues by combining multiple models to improve generalisation, robustness and accuracy [1][4-7]. However, class imbalance remains a major challenge, as defective modules typically form a small minority [10][11]. This skews predictions toward the majority class. Some of the common strategies to mitigate class imbalance and further enhance diversity include resampling (e.g. synthetic minority over-sampling technique (SMOTE), bootstrap, etc.), feature selection (FS), cost-sensitive learning, and threshold adjustment [9-11]. A typical ensemble-based SDP process is illustrated in Fig. 1. SDP begins with preprocessing raw code repositories, applying normalisation, noise filtering, and time-aware splitting, followed by feature extraction from static code, process, and change metrics. These features train tuned ensemble models, handle imbalance, evaluated using within-project (e.g., train–test splits) and cross-project (external datasets) validation. This pipeline supports accurate and generalisable defect prediction across varied software environments.

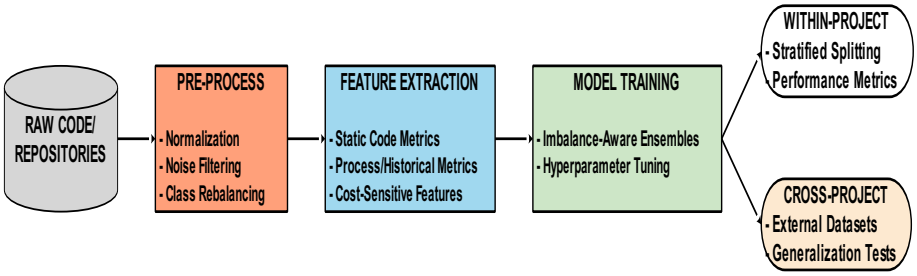

Figure 1. Ensemble-based SDP process

Particularly, EL helps deal with this issue by combining diverse base learners and flexible combination techniques. Its core idea is that a group of weak learners, when combined, can outperform a single strong model by reducing variance and overfitting. EL methods include bagging (bootstrap aggregating), boosting, stacking, and voting [1], [4-7]. Table 2 presents the taxonomy of ensemble methods for SDP. Bagging is widely used in prediction. It trains multiple base learners on different bootstrap samples, combining their predictions via majority voting or averaging to reduce variance and overfitting [4-7]. Bagging is effective for high-variance learners like DTs. Common algorithms include RF, Extra Trees (ETs), Balanced RF (BRF), Rotation Forest and Dynamic Ensemble Selection (DES). RF builds multiple trees using random feature subsets and bootstrap samples, handling high-dimensional data with minimal tuning. It improves the F1-score or area under the curve (AUC) by 5–15% in datasets like NASA and PROMISE, supports feature importance, and benefits from imbalance handling [4-7]. Extensions include adaptive sampling, dynamic feature weighting, and DL integration. ET, on the other hand, increases randomness via random split thresholds, training faster and suiting large, noisy datasets, but may underperform on smaller ones [4-7]. Other variants like BRF balanced sampling for class imbalance, Rotation Forest use principal component analysis (PCA) before bagging to capture feature correlations, and DES used instance-based classifier selection for real-time prediction [4-7].

Similarly, boosting sequentially trains weak learners, each focusing on the errors of the previous, producing a weighted final prediction [5],[6]. It focused on misclassified data and common algorithms, including AdaBoost and variants, Gradient Boosting (GBoost), and XGBoost. AdaBoost and its variants adjust instance weights to reduce bias and enhance minority class detection. For instance, Real and Gentle AdaBoost improve robustness to noisy labels, while multi-class versions handle severity levels [8]. Similarly, GBoost uses functional gradient descent on loss functions while XGBoost enhances error correction, efficiency, regularisation, and feature importance over RF [5-8]. LightGBM is known for fast training on large datasets with similar accuracy, and CatBoost natively handles categorical features [1]. Advanced methods like LogitBoost employ calibrated

probabilities via LR, BrownBoost is robust to outliers, and multi-objective boosting balances accuracy, false positives, and efficiency [7], [8].

Furthermore, the stacking ensemble uses a meta-classifier that learns from the base learners' predictions to make the final decision [1], [4-7]. This improves over simple voting by learning optimal combinations through cross-validation. In addition, linear meta-learners offer interpretability, while non-linear ones capture complex interactions but require more data [6]. Performance benefits from diverse learners and varied preprocessing. Advanced forms include multi-level stacking, dynamic selection based on local accuracy, blending for faster training, and Bayesian model averaging for probabilistic predictions with uncertainty estimates [5]. Voting-based ensembles combine predictions using majority vote or weighted voting, effective when classifiers are diverse but similarly accurate [4-7]. Consensus methods refine this by requiring full agreement (unanimous), flexible agreement (threshold-based/k-out-of-n), or weighting predictions by confidence, helping balance false positives and sensitivity in defect prediction [7].

By applying multiple perturbation approaches and combining various base learners, ensemble models effectively address class imbalance in SDP [10]. This approach improves accuracy, increases robustness, and offers competitive detection of defect-prone modules [10],[11]. Recent studies focus on advancing ensemble methods via hybrid strategies to enhance defect prediction by combining optimisation and model diversity. For instance, evolutionary ensembles use genetic algorithms (GAs) and swarm optimisation to tune model selection and parameters, with NSGA-II balancing performance and efficiency [4-7]. DL-based ensembles integrate varied neural nets or use dropout, with Deep Forests and CNN ensembles capturing structural code patterns [6], [7]. Lastly, heterogeneous ensembles mix different model types and feature views (e.g., static code, process, text), using techniques like PCA or independent component analysis (ICA) to expand and strengthen ensemble performance.

Table 2. Taxonomy of ensemble methods for SDP

Ensemble Type	Techniques
Bagging-based	- RF - Bootstrap Aggregating - Bagging with FS
Boosting-based	- AdaBoost - Gradient Boosting - XGBoost- LightGBM
Stacking	- Base Learners: DT, SVM, NB, etc. - Meta-Learner: LR, RF, XGBoost - Multi-layer Stacks
Voting-based	- Hard Voting

Ensemble Type	Techniques
	- Soft Voting - Weighted Voting
Hybrid and Novel Ensembles	- FS + Sampling + Ensemble - Optimisation + Ensemble (e.g., PSO, GA)-Clustering + Ensemble - DL Ensembles (CNN/RNN Stacks, Transformers) - Fuzzy & Rule-Based Ensembles (Choquet, Fuzzy Voting) - Cost-Sensitive Ensembles - Ranking-Oriented and Sequential Chains

2.3. Related Works

This section presents existing systematic reviews and surveys on ensemble methods in SDP, highlighting their insights into model performance, evaluation, and implementation. Table 3 summarises studies covering ensemble classifiers, hybrid models, FS, and class imbalance handling. It provides insights into common methods, strengths, and limitations, guiding the methodology and gap analysis of this review. Olivares-Galindo et al. [1] comparative analysis showed the practical advantages of ensemble classifiers over individual models, especially when paired with active learning. While this study proposed a new efficiency metric and broad setup, like much of the literature, it lacked DL and real-world validation. In the same way, authors in [5] and [6] highlighted the dominance of tree-based ensemble models, especially RF, Bagging, and Boosting, in SDP research. Both reported improved accuracy over baselines and widespread use of AUC and F-measure but provided mainly descriptive insights, with limited analysis of methodological trade-offs or robustness.

Similarly, authors [4] and [7] analysed ensemble approaches considering data quality, hyperparameter tuning, and study design. Particularly, [7] includes software change prediction (SCP), finding a high median AUC for ensembles, but limited exploration of stacking and SCP research. In another study, authors in [8] surveyed hybrid approaches combining search-based optimisation with machine learning, showing interest in improving model flexibility and accuracy. However, practical issues like industrial scalability and risk management receive limited attention. Beyond model design, reviews such as [9], [10], and [11] focused on processes such as FS and class imbalance handling. They find that ensembles work well with resampling methods like SMOTE and careful FS. However, comparisons of pre-processing strategies are uncommon, and context-specific guidance is limited.

Table 3. Summary of Related Works

Study	Scope	Key Contributions	Limitations
[1]	Ensemble vs Active Learning in SDP	Shows ensembles outperform single models; proposes EAPI metric; reduces training data significantly.	Focused only on classical ML; not tested in real-world environments
[5]	Review of ensemble classifiers	Highlights dominance of Bagging/Boosting; evaluates with AUC, F-measure	Lacks depth on data imbalance and computational cost
[6]	Ensemble learning in SDP	Covers preprocessing, tools, datasets; shows ensemble effectiveness	No performance benchmarks; broad recommendations
[4]	Critical review of ensemble ML (2018–2021)	Identifies class imbalance, tuning, and validation gaps	No meta-analysis or runtime discussion
[7]	SDP & SCP ensemble techniques	Finds tree-based ensembles most effective, stacking underwhelming	Overrepresents SDP; variable study designs
[8]	Hybrid search-based + ML techniques	Details GA-ANN, PSO-SVM use; discusses validation practices	Limited robustness analysis; few real-world datasets
[9]	FS in SDP	Categories filter and hybrid FS techniques; ensemble-friendly	Doesn't compare FS methods empirically
[10]	Imbalanced strategies in SFP	SMOTE + ensemble = best results; practical sampling strategies	No direct comparisons; lacks guidance by project type
[11]	Class imbalance learning in SDP	Highlights ensemble + sampling; GANs emerging	Descriptive; missing context-aware performance guidance

As shown in Table 3, across the studies, ensemble methods, particularly tree-based models like RF and bagging, are widely seen as effective for SDP [5 -7]. These often-outperformed single models and improved further with techniques like active learning [1] and sampling to address class imbalance [10],[11]. Some reviews provide broad overviews [5],[6], while others critically note challenges in tuning, validation, and practical use [4],[8]. In addition, specialised surveys cover FS [9] and hybrid optimisation [8], though few offer detailed context or empirical analysis. These studies show consensus on ensemble effectiveness and highlight the need

for more application-focused research. This paper contributes to existing research by analyzing the state-of-the-art in ensemble-based SDP and quantifying performance across studies and provides important research directions.

3. METHODS

The review conducted in this paper followed the PRISMA framework [12], [13] to ensure transparency and replicability. It focuses on advances in EL methods in SDP. A comprehensive search was systematically conducted across Scopus, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, Web of Science and Google Scholar, covering relevant peer-reviewed journal articles and conference papers from January 2020 to June 2025. Foundational studies published before and after this period were not included. Our search strings combined keywords such as "ensemble learning," "software defect prediction," "bagging," "boosting," "stacking," "voting," and names of specific algorithms (e.g., "random forest," "XGBoost") using Boolean operators. We iteratively refined the strategy based on initial yields and expert input. Supplementary searches included backwards and forward citation tracking of key articles, targeted scanning of top-tier conference proceedings, and manual inspection of relevant journal special issues.

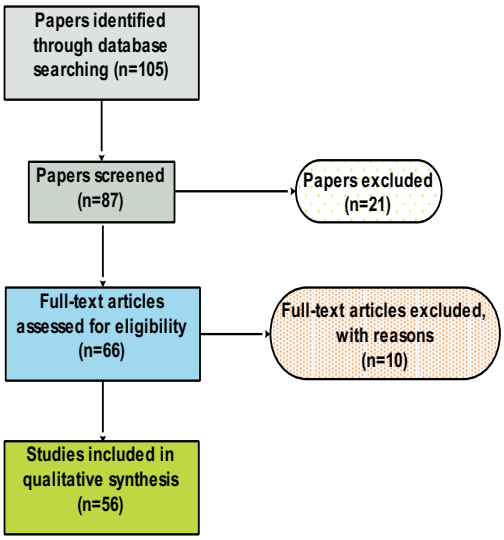

Figure 2. PRISMA process of selected studies

Following PRISMA guidelines, the identification phase yielded 105 records. After the removal of duplicates, titles and abstracts were screened for relevance using predefined inclusion criteria: (1) focus on SDP, (2) application of ensemble learning, (3) empirical evaluation with reproducible results, (4) publication in peer-

reviewed venues, and (5) sufficient methodological detail. Exclusion criteria eliminated non-ensemble SDP or purely theoretical studies, non-English papers, extended abstracts, duplicates, and those lacking empirical detail. Two independent reviewers performed the screening, with disagreements resolved by consensus. After a full-text review of 66 studies, 56 met the quality criteria. The included studies were assessed for methodological soundness, dataset appropriateness, evaluation rigour, and fairness in comparison. See Fig. 2. Data extraction covered ensemble configurations such as (base learners, combination techniques, hyperparameters), experimental setups (datasets, metrics, validation), performance metrics with statistical tests, and computational aspects. Results were standardised across metrics to support meaningful comparison, in line with PRISMA's emphasis on structured synthesis and reporting. In particular, single classifiers and ensemble methods were compared based on their effects on the standard performance metrics such as recall, F1-score, AUC, precision, accuracy, and MCC. The performance differences across metrics show the strengths and weaknesses of each strategy. In addition, since the effectiveness of ensemble models in handling class imbalance depends on dataset characteristics, the gains in the performance metrics were used to compare their impacts.

4. RESULTS AND DISCUSSION

This section presents the findings, which are the state-of-the-art ensemble-based SDP methods, their discussion, opportunity for future research and possible study weaknesses.

4.1. State-of-the-art Ensemble-based SDP Methods

This subsection presents the analysis of the ensemble-based SDP studies considered in this paper. We provided analysis by classifying the studies [14] – [69] into singular and hybrid strategies and summarising them in Tables 3a to 3f. The tabular summary of each study is based on its objective, methodology, datasets, and evaluation metrics used, as well as the results and critical assessment.

1) Hybrid ensemble-based

This subsection presents some of the studies that utilise different ensemble approaches such as stacking, boosting, bagging, voting, optimised and evolution, etc., to improve predictive accuracy. Jiang et al. [14] tackled two key challenges in SDP, high dimensionality and class imbalance, by introducing SMOTE-ERAR, a novel ensemble method. This approach combines multi-modal ensemble learning with Random Approximate Reduct (RAR) for attribute reduction and SMOTE for data balancing. Experimental results across 20 benchmark datasets reveal statistically significant gains over six existing techniques, particularly in F1-score,

Matthews Correlation Coefficient (MCC), and recall, with strong computational efficiency. Notable strengths include RAR's ability to preserve discriminative attributes and rigorous validation through statistical testing. However, reliance on discretisation may induce information loss, and the model's generalizability warrants further investigation. Similarly, Tong et al. [15] enhanced software defect number prediction (SDNP) by addressing extreme class imbalance with Subspace Hybrid Sampling Ensemble (SHSE), which integrates feature subspaces, a hybrid sampler (RusND + SMOTER), and weighted ensemble learning. Validated on 27 datasets, SHSE improves ranking accuracy by up to 15% and Fault-Percentile-Average (FPA) by 8.7–15.2% over several baselines. With large effect sizes (Cliff's δ: 0.48–0.668) and rapid testing time (0.0002s), SHSE is both efficient and practical for deployment. Nonetheless, its performance is sensitive to hyperparameters, and under-sampling may lead to data loss.

Table 3. Summary of Ensemble-based Defect Prediction

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[14]	Improve SDP under high dimensionality & imbalance	RAR-based reduction + SMOTE with voting ensemble	20 NASA & PROMISE datasets	F1, MCC, Recall, AUC	Avg. ↑ F1 4.2%, MCC 5.8%, Recall 6.1% ($p < 0.05$)	Efficient, imbalance-robust, limited to discretised attributes and datasets
[15]	Improve SDNP for imbalanced defect counts	Feature subspacing + SMOTER + RusND + weighted regression	27 public datasets (AEEEM, MetricsRepo)	FPA, RMSE, AAE	FPA ↑ 8.7–15.2%, large effect sizes	Broad dataset use; lacks interpretability; tuning-sensitive
[16]	Enhance ensemble diversity & class imbalance handling	NAR perturbation + SMOTE in multi-modal ensemble	20 datasets (NASA, Jureczko)	AUC, F1, MCC	AUC ↑ 3.09%, MCC ↑ 7.5%	Statistically strong; limited to three metrics
[17]	Improve WPDP with diverse classifiers	Bagging with 6 ML inducers + majority voting	43 datasets (PROMISE, NASA)	Accuracy, AUC	Accuracy ↑ 70.21%, AUC ↑ 62.5%; >88% projects improved	High coverage, interpretability, and runtime trade-off
[60]	Bug priority prediction via deep ensembles	5 CNNs + TF-IDF + majority voting	Bugzilla (25K+ reports)	Accuracy, F1, RMSE	F1 ↑ 73.12%, MAE ↓	CNN diversity is effective; limited to Bugzilla; compute-heavy

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[50]	Improve SDP via selective pruning	AUC-based pruning in a heterogeneous ensemble	8 PROMISE/Apache datasets	AUC, Accuracy, F1	AUC ↑ up to 4.4%; 7/8 wins	Pruning reduces noise, fixed thresholds, and the small model pool
[18]	Improve software quality prediction	Categorical encoding + ensemble learning (EL)	ISBSG, EBSPM, PROMISE	Accuracy	Accuracy ↑ 96.67%	Strong preprocessing; ignores imbalance; no runtime analysis
[19]	Compare ML, DL, EL on early imbalanced SDP	Benchmarking + SMOTE across models	22 PROMISE/NASA datasets	Accuracy, F1, G-Mean, AUC	AUC ↑ 44%, G-Mean ↑ 30%	Rich metrics; DL tuning overhead, limited deep architectures
[45]	Enhance CPDP with cost-aware ensemble	Cost-sensitive voting + TGF filtering	10 PROMISE datasets	AUC, Balanced Acc, F1	AUC = 0.632, F1 = 0.49	Stable across models; limited to Java projects; tuning gap
[48]	Joint handling of imbalance + dimensionality	SMOTE-ENN + K-PCA + RFE + XGBoost	NASA PROMISE (PC1–PC4, KC1)	Accuracy, F1, AUROC	F1 ↑ 97.2%, AUROC ↑ 99.3%	Efficient pipeline; scope limited to modular datasets

Jiang et al. [16] also proposed NRSEL, a neighbourhood rough sets-based ensemble learning method incorporating Neighbourhood Approximate Reduct (NAR). By perturbing both sample and attribute spaces and integrating SMOTE for minority oversampling, NRSEL addresses dimensionality and imbalance. Evaluation on 20 datasets shows significant improvements in AUC, F1, and MCC compared to seven ensemble baselines. Key strengths include effective attribute space perturbation and robust experimental design, although limited dataset diversity and narrow evaluation metrics temper generalizability. In parallel, Bhutamapuram and Sadam [17] introduced the Diverse Ensemble Learning Technique (DELT) to enhance within-project SDP through bootstrap aggregation and multi-inducer ensembles. Using NASA and PROMISE datasets, DELT outperforms single-inducer and baseline models in accuracy and AUC, supported by statistical significance. Its formal treatment of defect representation and empirical breadth are strengths, but added computational burden and interpretability concerns limit its adaptability to noisier, real-world data. Ceran et al. [18] explored ensemble-based prediction for software quality, transforming classification into a four-class problem and introducing novel categorical encoding. Through bagging, boosting, and stacking across three datasets, their approach

surpasses 96% accuracy in some cases. Strong pre-processing and methodical evaluation support these outcomes; however, practical deployment challenges, such as computational overhead and adapting to evolving data, are not addressed. Likewise, Dong et al. [19] evaluated ML, DL, and ensemble techniques under class-imbalanced conditions across 22 datasets, using metrics like AUC, G-Mean, and F1-score. Findings confirm EL's superiority, with improved balancing accuracy and runtime. The paper's comparative rigour and metric selection are strengths, though the limited investigation of advanced DL models and hyperparameter tuning challenges affects reproducibility.

Furthermore, Dong et al. [20] presented Double Fault Disagreement (DFD), a diversity measure to improve ensemble performance under imbalance. Their heterogeneous ensemble model showed consistent outperformance across ML, DL, and EL baselines on eight datasets. The method benefits from broad metric sensitivity and practical design, though concerns remain about dataset bias, overfitting, and unexplored ensemble frameworks. Also, Alsorory et al. [21] focused on class imbalance in object-oriented systems, proposing improved ensemble variants (E_BB, ROSBoost, E_EE) that leverage random oversampling and XGBoost. Tested on 19 PROMISE datasets, these models achieved significant gains in F-measure, balanced accuracy, and AUC. Their contribution lies in rigorous statistical testing and suitable metrics for imbalance, but limited dataset scope and small sample sizes necessitate broader validation. While Al-Fraiha et al. [22] explored enhancing SDP by combining EL models with hyperparameter optimisation using the NASA JM1 dataset. By comparing several ensemble methods (e.g., Bagging, AdaBoost, Voting, RF) to single learners and applying WEKA-based tuning, they found that optimised ensembles notably outperformed individual classifiers (e.g., voting accuracy rose to 0.8195). While offering valuable model selection insights, the study's generalizability is constrained by dataset duplication, imbalance, and reliance on a single source.

Khan [23] proposed the Hybrid Ensemble Learning Technique (HELT), integrating FS, k-means clustering, and EL methods (Bagging, AdaBoost) with base classifiers (SVM, NB, RF), along with preprocessing via discretisation and SMOTE. Across eight PROMISE datasets with 10-fold cross-validation, HELT achieved notable improvements in AUC, accuracy, recall, and F-measure, sometimes attaining perfect scores. Despite its methodological rigour, the study's external validity is limited by possible overfitting and limited exploration of alternative learners or sampling strategies. Similarly, Tong et al. [24] addressed multi-source cross-project defect prediction (MSCPDP) by proposing a weighted ensemble model informed by KL divergence, mutual information, and MIC for dataset, instance, and feature transfer. Evaluated on 30 defect datasets, the framework yielded significant performance gains, including up to 133.5% MCC improvement and better effort-aware metrics. Strengths lie in its transfer weighting

and multi-layered knowledge transfer, though scalability and generalisation beyond the tested contexts remain uncertain.

Moreover, Li et al. [25] tackled class imbalance and overlap via a two-stage ensemble combining adaptive neighbour cleaning, learning and cost-sensitive AdaBoost. Experimental results on the NASA and AEEEM datasets highlight gains in balanced accuracy and AUC, particularly for defective classes. However, performance variability, increased false positives, and unclear parameter-setting practices limit replicability across diverse datasets. Wan et al. [26] also presented SPE2, a self-paced ensemble framework using instance hardness to guide undersampling and mitigate overfitting. Through evaluation on 10 Java-based open-source datasets, SPE2_RF consistently outperformed baseline imbalance learners in F-measure, with low runtime costs. Yet, dependency on a single repository raises concerns regarding cross-project or industrial applicability. Likewise, Abbas et al. [27] developed ISDPS, a multi-layered SDP system leveraging data fusion, FS via FEFS, ensemble classifiers, and fuzzy logic fusion. Evaluated on a combined NASA dataset, ISDPS achieved 92.08% accuracy, surpassing standard ensembles. While its architectural layering and interpretability are strengths, its performance focuses on accuracy and historical data, limiting broader relevance and metric diversity. In the same way, Borandag [28] conducted a comparative study of ensemble ML and deep DL, introducing the SFP XP-TDD dataset and a novel hybrid RNN model. Using standard evaluation on three datasets, DL approaches, especially on large datasets, outperformed traditional ML, confirmed via ANOVA tests. While methodologically strong, the study's limited impact on smaller datasets and absence of cross-domain analysis temper its claims.

Alqarni and Aljamaan [29] suggested a GAN-boosted AdaBoost model to mitigate class imbalance in SDP. Through the integration of CWGANGP, CTGAN, and Bayesian optimisation, their model achieved superior MCC scores over traditional sampling (SMOTE, ROS, RUS) on 10 public datasets. However, its focus on boosting and limited examination of synthetic data quality suggests the need for broader architectural diversity and data validation. Dey et al. [30] also tackled high-dimensionality in SDP by combining a binary multi-objective starfish optimiser (BMOSFO) with a Choquet fuzzy integral-based ensemble classifier. Applied to five NASA datasets, the approach improved accuracy by 1–13% and reduced features by 33–57%, showing promise in dimensionality reduction and ensemble synergy. Still, limited scalability testing and reliance on modest computing infrastructure present practical constraints. Likewise, Tang et al. [31] introduced 2SSEBA, an ensemble that fuses Extreme Learning Machines with an enhanced sparrow search algorithm (2SSSA), integrating novel optimisation mechanisms such as pinhole imaging, reverse learning, and somersault foraging. Experiments on 25 public datasets demonstrated notable MCC gains and reduced false

positive/negative rates. Yet, the framework struggles with performance on small datasets, indicating challenges in sparse-data environments. In another study, Khlee et al. [32] handled class imbalance by combining homogeneous ensemble techniques (Bagging, Boosting, Voting) with near-miss under-sampling. Tested across several public datasets, the method achieved notable gains in recall and F-measure, key metrics in defect detection. Its strength lies in the diverse ensemble strategies and broad evaluation, though the absence of hybrid sampling or feature selection limits scalability and generalizability.

In the same vein, Srinivas et al. [33] presented a structured SDP framework that integrates minority oversampling (MOSD), adaptive sequential K-best (ASKB) FS, and a weighted RF (WRF) classifier. Applied to the NASA JM1 dataset, the model outperformed alternatives in accuracy and F1-score. Despite its effectiveness in addressing imbalance and feature relevance, reliance on a single dataset and limited analysis of scalability or computational cost restricts broader applicability. In addition, Alsawalqah et al. [34] proposed a segmented, heterogeneous ensemble classifier that uses K-means clustering to partition training data, assigning each segment its best-performing learner. Evaluated on 21 datasets, the model, particularly the KME variant, excelled in precision, recall, and G-mean compared to traditional and deep forest baselines. While statistically robust, its dependence on fixed clustering and K-means may overlook deeper data structures. Sha et al. [35] developed EL-NGDI, an ensemble framework that promotes base learner diversity through NGDI-driven feature perturbation and SMOTE for class balancing. Tested across 20 NASA and PROMISE datasets, EL-NGDI outperformed six ensemble baselines in AUC, F1, and MCC. Statistical validation supports these gains, yet untested hyperparameter stability and assumptions about oversampling conditions may hinder generalizability. Also, Habtemariam et al. [36] applied Bagging, AdaBoost, and Stacking to dual tasks: software reliability prediction (via MTBF) and defect classification. Using Musa and NASA datasets, the models achieved 94% prediction and 97% classification accuracy, surpassing prior benchmarks. The dual-task design adds value, though performance varies with hyperparameter tuning and depends heavily on standard datasets.

Dakuru and Kande [37] proposed a hybrid stacked ensemble that combines multiple base learners (e.g., Gaussian and Bernoulli NB, RF) with SMOTE for balancing and PCA for feature reduction, followed by XGBoost as a meta-learner. The approach achieved 84.6% accuracy, outperforming standalone models like XGBoost (81.2%). Despite its strong performance, inconsistencies in base learner reporting and minimal discussion of complexity or interpretability present limitations. Kiran and Ponnala [38] equally tackled class imbalance using a soft-voting ensemble of tuned LightGBM, CatBoost, and XGBoost, balanced with SMOTE. On the NASA JM1 dataset, their approach achieved 86% accuracy and 0.92 AUC, outperforming baselines. While the ensemble was effective in

leveraging G-mean and balancing strategies, the study's binary classification focus and narrow dataset scope limit broader implications. Gunda [39] applied boosting and voting ensembles to a large real-world dataset ("bugzilla.csv"), employing encoding and SMOTE for preprocessing. Gradient Boosting and LGBM achieved ~81.5% accuracy, though recall remained low, reflecting challenges in detecting minority-class defects. While robust in methodology, the consistently modest F1-scores and underwhelming recall indicate room for improved sensitivity. While Saheed et al. [40] introduced HSMOTE, a hybrid oversampling strategy incorporating ensemble classifiers (ET, RF, XGBoost) for severe class imbalance in NASA datasets. Results showed notable improvements in AUC, MCC, and F-measure compared to SMOTE + AdaBoost. Although effective at enhancing minority prediction, gains were modest in certain metrics, and the method lacks evaluation on DL approaches or broader datasets.

Table 4. Summary of Ensemble-based Defect Prediction

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[68]	Enhance SDP via ensemble FS	Filter ensemble with TOPSIS over Chi², IG, Relief, Correlation	PROMISE (KC3, MW1, PC1, PC2), Mylyn	nMCC, G-mean	KNN nMCC ↑ 8–16%, NB G-mean ↑ 14%	Imbalance-aware FS ensemble; limited classifiers and datasets
[20]	Improve ensemble diversity under imbalance	Double Fault Disagreement (DFD) + averaging ensemble	8 NASA/PROMISE datasets	AUC, MCC, F1, G-Mean	AUC ↑ 49.6%, MCC ↑ 298%	Strong diversity gains; ensemble fusion scope limited
[51]	Improve accuracy via resampling + FS ensemble	WCP-SMOTE + GDE + RSM + voting	10 PROMISE datasets	Accuracy, F1, AUC	F1 gains in 7/8 datasets	Hybrid sampling & FS; limited sampling variety
[21]	Boost OO defect prediction	Enhanced Boosting (E_EE) + ROS + XGBoost	19 PROMISE OO datasets	F1, AUC, Balanced Acc	AUC ↑ 0.95, F1 ↑ 0.84→0.89	Validated results: OO-only and small datasets
[62]	Defect ranking with ensemble SVM	Pairwise Ranking SVM + kernel mapping	20 OSS projects	FPA, P20, Kendall's tau	FPA ↑ 22%, P20 ↑ 34%	Broad eval, scalable; limited OSS scope, kernel tuning
[22]	Tune ensemble learners in SDP	Bagging, AdaBoost, Voting tuned via WEKA	NASA JM1	Accuracy, ROC, Kappa, F1	Accuracy ↑ 0.72→0.82	Practical tuning insights: lacks balancing,

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
						narrow dataset
[23]	Hybrid EL via FS and clustering	GA FS + k-means + AdaBoost + Bagging	8 PROMISE datasets	Accuracy, AUC, Recall, F1	SVM-based ensemble Accuracy & AUC ↑ 100%	Strong validation: diversity limited, overfitting risk
[24]	Improve CPDP via multi-source transfer	Transfer learning + MI-based ensemble weighting	30 datasets (AEEEM, ReLink, JIRA)	AUC, MCC, Popt20%, IFA	MCC ↑ 133%, IFA ↓ 48%, AUC ↑ 2.6%	Effort-aware and scalable; costly, dataset scope finite
[52]	Classify bug types using ensembles	TF-IDF + augmentation + RF, NB, LR, SVM voting	~2,000 reports (Mozilla, Eclipse)	Accuracy, Precision, Recall, F1	Accuracy ↑ 96.7%	Novel augmentation: small scale, lacks runtime analysis
[25]	Address imbalance & overlap in SDP	NCL + cost-sensitive AdaBoost + Bagging	NASA, AEEEM	AUC, Balanced Acc, TPR	Balanced Acc ↑ 7%, AUC ↑ 9.5%	Dual challenge handled; FPR increase, tuning unclear

2) Stacking ensemble-based

This subsection presents the studies that utilised only the stacking ensemble strategy to improve SDP. Alazba and Aljamaan [41] explored the impact of hyperparameter optimisation on ensemble-based SDP, focusing on stacking generalisation. Using grid search across seven tree-based ensembles and a meta-classifier for stacking, their method was tested on 21 defect datasets with multiple metrics. Results consistently favoured the stacked ensemble, which outperformed all individually tuned models in AUC and F-measure. Strengths include rigorous experimentation with repeated cross-validation and statistical testing, enhancing result reliability. However, the lack of class balancing and the unquantified computational overhead of tuning complex models like CatBoost limit practical scalability. Similarly, Islam et al. [42] proposed a two-layer stacking ensemble using static code metrics, combining base classifiers (RF, XGB, SVM, KNN, LR) with LR as the meta-learner. To address class imbalance, SMOTE was incorporated. Evaluated on eight Apache projects and a real-world Python dataset, the model improved accuracy (e.g., Ant: from 82.3% to 89.3%, Xerces: from 77.1% to 90.8%) and yielded an AUC of 0.95. Its effectiveness lies in attention to overfitting and empirical validation beyond benchmarks, though dependence on Python tools and limited scalability and logical expression handling present challenges.

Furthermore, Wicaksono et al. [43] introduced an ensemble stacking framework that integrates transformer-based models, CodeBERT and CodeGPT, with data augmentation to mitigate class imbalance. Using the PROMISE dataset, the ensemble achieved modest but consistent improvements (1–3%) in accuracy, F1-score, precision, and recall. The model effectively combines the syntactic and semantic strengths of pre-trained DL models and applies a clear ensemble strategy. Nonetheless, reliance on a single Java-based dataset and limited performance gains reduce its immediate applicability to industrial-scale use cases. Likewise, Muhammad et al. [44] tackled class imbalance through stacking ensemble learning and random oversampling. Their model combines base classifiers (RF, SVM, XGBoost, KNN) with LR as a meta-classifier and includes preprocessing steps such as outlier filtering and scaling. Evaluated on four PROMISE datasets, results show strong accuracy and F1 performance, exceeding 90%, with random oversampling outperforming both SMOTE and under-sampling. While effective in mitigating imbalance, the model's use of classical learners and lack of testing beyond standard datasets constrain its generalizability to broader software contexts.

3) Bagging ensemble-based

A few studies that utilised only the bagging strategy to improve SDP are discussed in this subsection. Li et al. [45] proposed the Cost-Cognitive Bagging Ensemble (CCBE) to address both class imbalance and project heterogeneity in cross-project defect prediction (CPDP). The framework includes balanced resampling, base classifier training, automated cost value cognition, and ensemble voting. Evaluated on 10 PROMISE datasets with varied imbalance levels, CCBE consistently outperformed traditional classifiers and ensembles in AUC, balanced accuracy, and F1-score, reflecting its stability and adaptability to imbalanced CPDP scenarios. Nonetheless, limitations stem from its focus on medium-sized Java projects and a relatively narrow benchmark set, which may constrain generalizability. Similarly, Zhang et al. [46] enhanced just-in-time SDP (JIT-SDP) by combining ensemble learning with four oversampling techniques to mitigate data imbalance and improve model robustness. Their systematic exploration paired oversamplers with four ensemble algorithms across several datasets, identifying RandomOverSampler with RF as the most effective configuration. Notably, an ensemble size of 15 models balanced predictive performance and runtime cost. The study's strength lies in its comprehensive empirical comparisons and parameter analysis, though concerns remain around overfitting from oversampling and adaptability to dynamic software environments due to static dataset reliance. In another study, Samantaray and Das [47] evaluated the bagging ensemble's impact on SDP performance using nine NASA PROMISE datasets. Their approach incorporated SMOTE for balancing and feature scaling for improved learning, benchmarking several base learners with and without Bagging, including RFC as a reference.

Results revealed that Bagging consistently improved classifier accuracy, with RFC reaching up to 98.7%. The study demonstrates bagging's effectiveness, strengthened by multi-dataset testing and detailed pre-processing. However, its narrow focus on accuracy, lack of computational or interpretability analysis, and absence of statistical validation limit broader insights.

Table 5. Summary of Ensemble-based Defect Prediction

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[53]	Unsupervised SDP without labels	Sparse PCA + Chi² FS + clustering ensemble (Spectral, Newman, Fluid, CNM)	PROMISE, NASA (16 projects)	AUC, F1	AUC ↑ 7.7–15.4% (PROMISE), 1.4–5.0% (NASA)	Label-free and interpretable FS; metric sensitivity, heuristic labelling
[26]	Handle imbalance and data scarcity	Self-paced ensemble undersampling via instance hardness	10 PROMISE Java datasets (SeaCraft)	Precision, Recall, F1	Recall ↑ 14.3%, F1 ↑ 4.2%; runtime < 1.5s	Efficient, robust tuning; lacks CPDP, OSS-only
[54]	Improve SBP via instance-wise weighting	Reward-based voting + ROSE resampling	28 datasets (SOFTLAB, NASA, ReLink, etc.)	Accuracy, F1, MCC	Accuracy ↑ 19.5%, MCC ↑ 0.42	Broad validation and balance-aware; binary only, industry scope limited
[55]	Imbalance-robust SDP via meta-stacking	DNN stacking over 10-fold weighted ensembles	8 NASA MDP datasets	AUC, F1, MCC	F1 ↑ 4.2%, MCC ↑ 5.1% (significant)	Diverse learners; lack feature/code-based diversity
[56]	Improve ensemble accuracy via tuning	Tuned RF, SVM, NB, ANN + voting	NASA MDP (7 projects)	Accuracy, PPV, TNR, MR, FNR	Accuracy ↑ 92.16% (PC1); wins all datasets	Strong optimisation; dated datasets, narrow metric focus
[27]	Early prediction via fuzzy ensemble fusion	FEFS FS + Bagging/Voting/Stacking + fuzzy decision layer	NASA MDP (5 datasets)	Accuracy, Miss Rate	Accuracy ↑ 92.1%, Miss ↓ 7.9%	Interpretable fusion; few metrics, legacy data
[41]	Evaluate tuned TE stacking	Stratified 10×10 CV on 7 tree models + stacking meta-learner	21 public datasets	F1, AUC, Brier	Stacking wins on 80% datasets	Robust eval; lacks imbalance handling, tuning cost ignored
[28]	Compare DL and ensemble ML for SFP	CNN, LSTM, RNNBDL vs. RF, Rotation Forest	SFP (XP-TDD,	Accuracy, AUC, F1, Kappa	RNNBDL ↑ Accuracy	Public dataset release; scope

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
			Eclipse, Active MQ)		95.9%, RF ↑ 5.8%	limited to in-project
[29]	GAN-enhanced boosting for imbalance	CTGAN/CWGANGP + AdaBoost + Bayesian tuning	10 public SDP datasets	MCC	Outperforms SMOTE, ROS; undersampling harms GANs	Strong validation of tuned GANs; quality of synthetic data not evaluated

4) Boosting ensemble-based

Like bagging, a few studies, Mustaqeem et al. [48], enhanced SDP by dealing with common challenges such as imbalanced datasets and high feature dimensionality. The technique combines data balancing (SMOTE-ENN), FS (RFE-CV), dimensionality reduction (K-PCA), and an XGBoost classifier with hyperparameter tuning. Empirical evaluation on multiple NASA PROMISE datasets showed statistically significant improvements in accuracy, F1-score, and AU-ROC compared to existing techniques, while maintaining computational efficiency. However, the study's generalizability beyond module-level defect detection and to other domains remains untested, and practical integration into real-world continuous deployment pipelines is not addressed. In the same way, Dar and Farooq [49] simultaneously tackled two prominent challenges in real-world SDP: dataset class imbalance and class overlap. The technique uses a four-stage pipeline: data cleaning, NCL for overlap removal, RUS for handling imbalance, and XGBoost for classification, combining data- and algorithm-level strategies within an ensemble framework. Evaluation results on 16 PROMISE datasets show steady improvements in recall, G-mean, F-measure, and AUC over 10 baselines, supported by difficult statistical testing. The approach includes a comprehensive approach to class imbalance and overlap, and strong metrics, but is limited by computational cost, potential information loss from random under-sampling, and unclear applicability to highly imbalanced or noisy industrial data.

Table 6. Summary of Ensemble-based Defect Prediction

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[30]	Accuracy and efficiency on high-dimensional SDP	BMOSFO + Choquet fuzzy ensemble; multi-objective FS	NASA PROMISE (JM1, KC1, PC1, KC2, CM1)	Accuracy, Precision, F1, Wilcoxon, Hypervolume	Accuracy ↑ 1–13%, FS ↓ 33–57%, final ↑ 95%	Interpretable FS and compact model; lacks modern systems
[32]	Enhance accuracy and	IECGA: GA-based FS + RF, SVM, NB voting	7 NASA datasets	Accuracy, AUC, F1, Precision	Accuracy ↑ 95.1% (PC1),	Strong integration; NASA-only

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
	runtime via FS + EL				runtime ↓ 51%	scope, limited baselines
[37]	Stabilise ensemble models via optimisation	2SSSA-tuned bagging ELMs with novel foraging strategy	25 datasets (NASA, SOFTLAB, AEEEM, etc.)	MCC, G-mean, FNR, FPR	MCC ↑ 0.75 avg; stable convergence	Wide evaluation; small-feature sets underperform
[38]	Fast and accurate bug prediction	PCA + LR + Extra Tree with vectorisation	NASA PC5	Accuracy, F1, AUC, Recall	Accuracy ↑ 97.8%, F1 ↑ 99.7%, runtime ↓ 27%	High speed and accuracy; generalizability unexplored
[39]	Tuning + imbalance handling for SDP	AVSEB: Bagging ELMs + AVSSA + cut-point prep	15 datasets (NASA, JIRA, MORPH)	Recall, MCC, G-mean, F1	Recall ↑ to 1.0, G-mean ↑ 0.91	Joint tuning/resampling; high runtime
[65]	Boost accuracy with hybrid GP + boosting	GP for features + AdaBoost	NASA PROMISE (KC1–LUCENE)	Accuracy, AUC	Accuracy ↑ 97.4% (KC1), AUC = 0.91 ± 0.02	Diverse tests and pipelines; compute-intensive
[49]	Handle imbalance + overlap	Data cleaning → NCL → RUS → XGBoost	16 PROMISE datasets	Recall, AUC, G-mean, F1	Recall ↑ 85%, AUC ↑ 76.2%, F1 ↑ 40%	Multi-stage and robust; RUS risk, sparse compute details
[32]	Boost prediction via undersampling + EL	Bagging/Boosting/Voting + Near Miss	PROMISE, Eclipse, GitHub, BugCatchers	AUROC, Accuracy, F1, MCC	AUROC ↑ 98%, Accuracy ↑ 94%	Multi-dataset success; risk of losing key samples
[33]	Tackle imbalance + FS with pipeline	MOSD + ASKB FS + Weighted RF	NASA JM1	Accuracy, F1, Precision, Recall	Accuracy ↑ 99.1%, F1 ↑ 99.3%	Detailed pipeline, ablation-backed; lacks scalability test
[34]	Exploit heterogeneity via cluster-wise EL	K-means → cluster-specific EL	21 public datasets	G-mean, Recall, Precision	Outperforms 19/21; precision ↑	Adaptive to structure; fixed k constraint

5) Voting ensemble-based

Several studies that employed voting strategies in enhancing SDP are discussed in this subsection. Agrawalla and Reddy [50] suggested an Optimal Classifier Selection (OCS) algorithm aimed at refining heterogeneous ensembles by pruning base classifiers that underperform relative to an AUC threshold. Applied across eight benchmark datasets using the average of probabilities and majority voting as ensemble rules, the method showed measurable gains in AUC (up to +4.4%),

accuracy, and F-measure compared to traditional voting ensembles. The strength of the study lies in its systematic pruning strategy and thorough threshold exploration, which reduces redundancy in ensemble construction. However, its reliance on manually set thresholds and the restricted pool of classifiers limits scalability and the examination of broader ensemble configurations or class imbalance strategies. Similarly, Wei et al. [51] introduced E_RHFS, an ensemble method addressing class imbalance and classifier diversity. It combines WCP-SMOTE, a complexity-weighted oversampling method, and a hybrid feature selector combining granular decision entropy with random subspace techniques. Tested on 10 benchmark datasets using KNN and CART as base classifiers, the model achieved consistent improvements in accuracy, F1-measure, and AUC over existing ensemble approaches. Notable strengths include the intelligent fusion of sampling and FS. Nonetheless, the study's emphasis on standard datasets and limited variety in sampling methods highlights areas for future expansion.

Alsaedi et al. [52] also focused on bug report classification by leveraging a voting ensemble of traditional classifiers (RF, LR, Multinomial NB, and SVM) enhanced through text augmentation. Using a public dataset of ~2,000 bug reports, the ensemble's accuracy rose from 90.42% to 96.72% with augmentation, affirming the benefits of synthetic data expansion. The work stands out for its creative use of NLP techniques and comprehensive baseline comparisons. Yet, its domain-specific dataset and absence of deeper evaluations on class imbalance or processing overhead may affect real-world adaptability. Likewise, Tao et al. [53] proposed an unsupervised SDP framework to address the scarcity of labelled data, combining chi-squared sparse FS with ensemble clustering. By integrating sparse PCA with statistical filtering and aggregating multiple clustering algorithms via hard voting, the method achieved superior performance over baseline clustering models on PROMISE and NASA datasets. Its efficacy in label-scarce contexts and innovative use of adjacency matrices is commendable. However, sensitivity to dataset size and imbalance, coupled with inconsistent performance against other dimensionality reduction techniques, temper its reliability. Kumar and Chaturvedi [54] equally developed a reward-based Weighted Majority Voting (WMV) ensemble model that dynamically assigns weights to classifiers based on their predictive correctness. Assessed on 28 datasets with extensive statistical testing, the method consistently outperformed individual learners, simple majority voting, and several state-of-the-art baselines. The integration of ROSE sampling also enhanced F-measure and MCC scores. While WMV addresses fixed-weight limitations and supports class imbalance handling, its efficacy depends on base classifier diversity and is currently confined to binary classification contexts with metric-specific data.

Furthermore, Chen et al. [55] suggested a dual-layer ensemble model to address class imbalance in SDP, incorporating weighted homogeneous ensembles with heterogeneous stacking and a neural network meta-classifier. The model includes

an effective FS process and was evaluated on eight NASA MDP datasets. Results show consistent gains in F1 and MCC scores over baseline models. The approach stands out for its thorough analysis of classifier diversity and an ablation study validating the neural network's contribution. However, its sole reliance on NASA datasets, known for quality concerns, limits generalizability to contemporary software environments. In parallel, Ali et al. [56] introduced VESDP, a two-stage ensemble model employing four heterogeneous ML classifiers integrated through a voting mechanism. The pipeline includes iterative parameter tuning and structured training/testing procedures. Experiments on seven NASA MDP datasets reveal significant accuracy improvements over twenty baseline methods, particularly on datasets such as PC1 and MW1. While the systematic tuning and ensemble synergy enhance predictive performance, the study is constrained by its heavy reliance on historical datasets and by its evaluation emphasis on accuracy, neglecting broader performance metrics.

Ali et al. [57] also proposed IECGA, a five-stage framework for SDP combining GA-based FS, heterogeneous classifiers, and soft voting. The pipeline encompasses dataset preprocessing, dimensionality reduction, classifier optimisation, and ensemble integration. Tested on seven NASA datasets, IECGA demonstrated accuracy improvements up to 95.1% and over 50% reductions in training and inference time. Its strength lies in balanced FS and diverse classifier fusion, though limitations include the focus on accuracy, lack of alternative method comparisons, and dependence on older benchmark datasets. Likewise, Mehta et al. [58] applied a voting ensemble, comprising GNB, BNB, RF, and SVM, coupled with PCA for dimensionality reduction to improve SDP. Supported by Condorcet's Jury Theorem, the study aimed to overcome individual classifier limitations. Achieving a competitive accuracy of ~95.7% on the CM1 dataset, the model emphasises interpretability and theoretical grounding. However, it is hampered by reliance on a single dataset, lack of evaluation depth, absence of class imbalance consideration, and speculative biomedical application claims lacking empirical validation. Joy et al. [59] explored a soft-voting ensemble combining DT, RF, GB, and SVM classifiers, evaluated using NASA PROMISE datasets. The workflow included preprocessing, base learner application, and confusion-matrix-based performance evaluation. While the ensemble showed consistent improvements in accuracy and F1 score, especially in identifying non-defective classes, it struggled with defective class detection due to the absence of imbalance handling strategies. Despite being practically accessible, the model's lack of advanced ensemble mechanisms like stacking or bagging limits its methodological innovation.

Table 7. Summary of Ensemble-based Defect Prediction

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[58]	Improve SDP using ensemble classification	PCA + CJT-based voting over GNB, BNB, RF, SVM	PROMISE CM1	Accuracy, Precision, Recall	Accuracy ↑ 95.67%, Precision ↑ 94.37%, Recall ↑ 95.55%	Interpretable ensemble; single dataset, no imbalance handling
[35]	Improve accuracy and robustness	EL-NGDI: NGDI perturbation + SMOTE + ensemble	20 NASA & PROMISE datasets	AUC, F1, MCC	AUC ↑ 0.9225, F1 ↑ 0.9201, MCC ↑ 0.8420	Strong across metrics; tuning complexity (σ, δ)
[36]	Predict failure time + defect labels	Dual-task ensemble (regression + classification)	Musa, NASA	MTBF error, Accuracy	Failure prediction ↑ 94%, classification ↑ 97%	Versatile task integration; benchmark-only, tuning-sensitive
[59]	Compare ensemble vs individual classifiers	Soft voting over DT, RF, GB, SVM	NASA PROMISE (5 projects)	Accuracy, F1, Precision, Recall	Accuracy ↑ 94.6%, F1 ↑ 0.90–0.97 (non-defect)	Simple yet robust; ignores imbalance, weak defect recall
[42]	Early SDP with imbalance handling	Stacked EL (base + LR meta) + SMOTE	8 Apache + 1 real Python project	Accuracy, F1, AUC	Accuracy ↑ to 90.8%, AUC ↑ 0.95	Balances data + architecture; Python-only, no semantic info
[9]	DL ensemble for defect prediction	PCA + CNN, MLP, LSTM ensemble	Proprietary (~170K instances)	Accuracy, F1	Accuracy ↑ 78% (ensemble), CNN ↑ 87%	DL diversity, good F1; interpretability and stats weak
[69]	Enhance FS via ensemble evaluation	RF-embedded FS using C-SU (Correlation + SU)	11 NASA MDP + 8 OSS	AUC, F1, Precision	AUC ↑ 13.06%, F1 ↑ 12.56%, Precision ↑ 91.4%	Efficient FS + broad validation; modest dataset spread
[66]	Rank defects by severity	GBRank + MLP + monotonic projection on AST/static	PROMISE (16 versions)	FPA, PD20	FPA ↑ 0.742, PD20 ↑ 0.69	Semantic-aware ranking, complexity, and a single dataset limit
[46]	Improve JIT-SDP under imbalance	OSNECL: Grid-tuned 4 oversamplers	11 projects (OSS + commercial)	Accuracy, F1, Recall, Precision	RF+ROS: Accuracy ↑ 97%, F1 ↑ 0.93	Extensive tuning gains; overfitting +

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
		× 4 EL classifiers				adaptivity unexplored

6) Deep learning ensemble-based

In addition to the transformer approaches in [43], Dharmakeerthi et al. [60] addressed the automation of bug priority prediction through a deep ensemble framework combining multiple CNN architectures aggregated via majority voting. The model includes comprehensive pre-processing and diverse feature extraction methods, evaluated on a large-scale Bugzilla dataset. Results show the ensemble outperforms individual CNNs in precision, recall, F-measure, and error metrics, aligning closely with expert judgment. However, the reported ensemble accuracy of 79.18% contrasts with higher accuracies from individual models, potentially due to differences in metric aggregation or decision fusion. Key strengths include rigorous evaluation and architectural diversity, though high computational costs, limited analysis of ensemble diversity, and the sole use of Bugzilla data constrain broader applicability. Similarly, Charan et al. [61] proposed an ensemble of DL models, MLP, CNN, and LSTM, designed to increase predictive accuracy. Their methodology includes dimensionality reduction via PCA and systematic evaluation of each model before integration. Applied to an imbalanced dataset, the ensemble achieved an accuracy of 0.78, outperforming individual models overall, though CNN alone exhibited higher accuracy in specific cases. While the approach effectively explores DL architectures and addresses interpretability, it lacks nuanced handling of imbalance, omits statistical significance testing, and provides limited comparative analysis with conventional machine learning baselines.

Table 3f. Summary of Ensemble-based Defect Prediction

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[37]	Enhance prediction via class-balanced stacking	SMOTE + PCA + GNB/BNB/RF stack → XGBoost meta	NASA, PROMISE	Accuracy, Precision, Recall, F1, AUC	Accuracy ↑ 3.4%, F1 ↑ 81.6%	Balanced stack; dimensionality reduction; base learners unspecified; lacks cost analysis.
[38]	Address imbalance via boosting fusion	SMOTE + tuned LightGBM, CatBoost, XGBoost	NASA JM1	Accuracy, AUC, G-mean, F1	Accuracy ↑ 24%, AUC ↑ 22%, G-mean ↑ 26%	Boosting synergy, solid improvements; limited to one dataset, lacks complexity analysis.

Study	Objective	Methodology & Innovation	Datasets	Metrics	Results	Critical Assessment
[47]	Assess Bagging across classifiers	SMOTE + Bagging over 5 base learners	9 NASA datasets	Accuracy	RFC ↑ to 98.8% (PC2); Bagging helped most	Simple setup; broad base models, accuracy-only view; no stat testing
[67]	Build an intelligent ensemble with FS	Optimized RF/SVM/NB/MLP + Stacking (DT, RF, LGBM) + SMOTE + PSO FS	NASA (7 projects)	Accuracy, Precision, Recall	Accuracy ↑ 93.5% (PC4)	PSO-guided FS; balanced pipeline; interpretability unexplored
[39]	Benchmark ensembles on real-world data	SMOTE + encoding + ensemble comparison	Bugzilla (~170K reports)	Accuracy, Precision, Recall, F1, AUC	GB ↑ 81.5%, AUC ↑ 0.7887	Real-world validation; low Recall (0.39); no deep feature analysis
[43]	Use transformer ensembles for SDP	CodeBERT + CodeGPT + augmentation	PROMISE (14 Java projects)	Accuracy, F1, Recall, Precision	Accuracy ↑ 0.9%, F1 ↑ 1.5%	Semantic and syntactic fusion; Java-only; modest improvement
[44]	Improve stacking with oversampling	RF/SVM/XGBoost/KNN stack + LR meta + SMOTE	PROMISE (KC1, KC2, PC1, PC3)	Accuracy, F1, AUC	Accuracy ↑ 99.1%, F1 ↑ 99.2%	High performance: strong validation, limited to classical ML
[40]	Enhance minority defect detection	HSMOTE + Extra Tree/RF/XGBoost + normalisation	NASA (JM1, KC1, PC3)	Accuracy, AUC, MCC, F1	Accuracy ↑ 81–94%, AUC ↑ 73–85%	Balanced ensemble; HSMOTE integration; no DL; NASA-only

7) Optimised and evolutionary ensemble-based

This section presents studies that employed optimisation and evolutionary strategies to enhance SDP. Yang et al. [62] tackled software defect ranking by proposing a nonlinear Ranking SVM (RSVM) approach augmented with kernel mapping and ensemble learning to optimise testing resource allocation. By combining nonlinear transformations, an enhanced SMO algorithm, and ensemble-based learning, the model addresses computational demands inherent to pairwise ranking tasks. Evaluation across 20 open-source datasets revealed significant improvements in defect prioritisation metrics,22.22% in FPA and 34.55% in P20, compared to baseline RSVM models, all while maintaining competitive execution times. Strengths include the fusion of nonlinear modelling with computational efficiency and a rigorous empirical framework. However,

limitations stem from the exclusive use of open-source datasets and the need for deeper exploration of kernel choice and class imbalance effects. Equally, Johnson et al. [63] suggested an optimised ensemble for SDP that combines LR and ETs classifiers, with dimensionality reduction handled by vectorisation and PCA. The approach was validated using the NASA PC5 dataset and tested across WEKA, MATLAB, and PyCharm platforms. Achieving 97.8% accuracy, the model outperformed baseline classifiers while minimising prediction time and memory usage. The study's strengths include cross-platform validation, efficient FS, and a clear experimental setup. However, challenges such as overfitting, limited dataset diversity, and trade-offs between runtime and memory efficiency remain significant limitations.

Tang et al. [64] also tackled parameter optimisation and class imbalance by developing the Adaptive Variable Sparrow Search Ensemble Bagging (AVSEB) model. It integrates an Extreme Learning Machine predictor with AVSEB for parameter tuning, alongside bagging and an unstable cut-point algorithm to manage imbalance. Evaluation on 15 open-source datasets showed substantial improvements across recall, G-mean, F-measure, and MCC, supported by significance testing. The method excels in accuracy and stability, yet its algorithmic complexity and elevated computational cost may hinder deployment in resource-constrained settings. Likewise, Sahu et al. [65] designed a hybrid SFP model that combines Genetic Programming (GP)-based feature generation and selection with AdaBoost ensemble learning. Using GP for evolving relevant features, followed by refinement and classification through AdaBoost, the model achieved improved accuracy and AUC across several NASA PROMISE datasets relative to statistical and machine learning baselines. Its strengths include adaptability to heterogeneous data and robust performance on complex datasets, but the stochastic nature of GP introduces variability in results and high computational demands.

Zhao et al. [66] introduced SeDI, a semantics-enhanced ensemble for Ranking-Oriented SDP (ROSDP), designed to prioritise defect severity over binary classification. SeDI integrates static software metrics with semantic features derived from abstract syntax trees using a hierarchical attention network (PHAN), combines outputs from GBRank, and applies a meta-learner MLP through a monotonic projection function. Grounded in learning-to-rank principles, it utilises hinge loss and pairwise comparisons. Experiments on the PROMISE dataset showed SeDI outperformed RankNet, RSVM variants, and DPNN in FPA and PD20 metrics. Strengths include semantic feature modelling and ranking fidelity, though reliance on a single dataset and computational overhead from combining GBRank with MLP present scalability concerns. In the same vein, Tamilkodi et al. [67] proposed a two-stage intelligent ensemble for SDP that combines Adaptive Voting and Stacking classifiers with individual classifier optimisation. The method includes preprocessing (data cleaning), class imbalance correction via SMOTE, and

FS through PSO. Evaluation on seven NASA datasets showed the Stacking classifier consistently outperformed individual models and Adaptive Voting, achieving up to 93.5% accuracy. Its strengths lie in optimisation techniques and class imbalance handling. However, the absence of deeper performance metrics like F1 or AUC and limited discussion on interpretability and computational cost constrain a comprehensive evaluation.

8) Ensemble filter-based feature selection

This subsection presents studies that combined an ensemble method with FS strategies. Kaur and A. Kumar [68] proposed MCDM-EFS, an ensemble filter-based FS technique that utilises Multi-Criteria Decision Making (MCDM) through the TOPSIS method to improve SDP. By aggregating the outputs of multiple filter-based selectors into a decision matrix, the approach addresses the limitations of individual feature ranking methods. Tested on five imbalanced datasets using KNN and NB classifiers, the model achieved statistically significant improvements in normalised MCC and G-measure over standalone selectors and no-selection baselines. Strengths include its systematic integration strategy, treatment of class imbalance, and robust validation. However, its reliance on only two classifiers and datasets from well-known repositories limits scalability and applicability across broader contexts. Similarly, Li et al. [69] introduced C-SU, an ensemble FS approach that integrates Correlation-based Feature Selection (CFS) and Symmetric Uncertainty (SU) within an RF classification framework to manage high-dimensional SDP datasets. The method iteratively refines feature subsets based on classification performance, aiming to retain predictive attributes while reducing redundancy. Empirical testing on NASA and open-source datasets demonstrated significant gains in F1-score, Precision, and AUC compared to baseline FS methods, supported by statistical validation. While effective in improving model quality and efficiency, the method's iterative weighting design may not scale efficiently to very high-dimensional settings, and its focus on a limited set of datasets constrains generalizability.

4.2. Discussion

This paper provides a comprehensive survey of ensemble techniques applied to SDP in the development lifecycle. This section presents the findings discussion of the findings. Firstly, our review considered fifty-six relevant papers published from January 2020 to June 2025. As shown in Fig. 3, the publication trend over the last six years reveals a steady and intensifying research interest in ensemble learning for SDP. The field started slowly in 2020 and 2021with only three papers collectively, picked up pace in 2022 and surged in 2023. The year 2024 marked a significant peak, contributing 21 papers: 13 journals, 8 conferences, the highest volume in the dataset. While 2025 shows a slight decline, it still maintains substantial output with

14 publications, especially considering the year is only halfway through. Overall, journal contributions consistently outweigh conference proceedings: 37 vs. 19, indicating a steady focus on more in-depth, sustained research in the field. This progression signals growing academic interest, along with a stronger push toward practical use and greater methodological rigour in ensemble-based SDP.

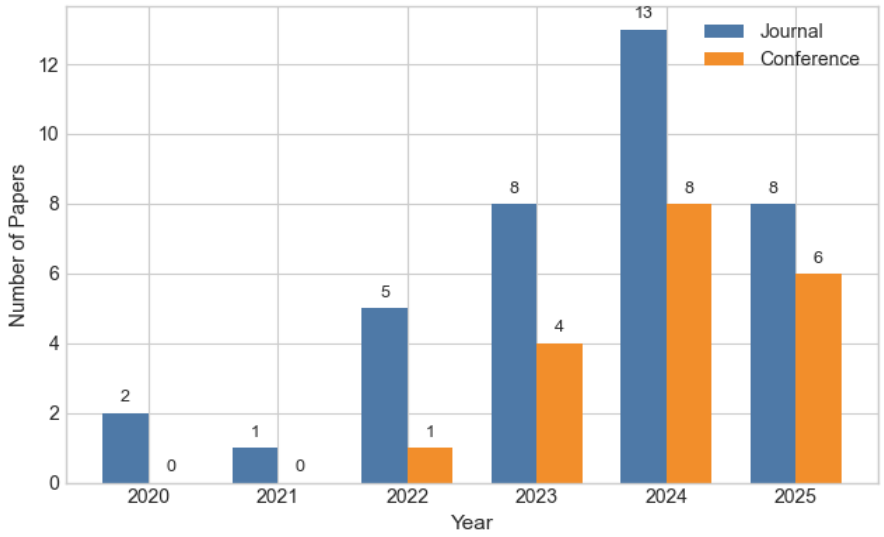

Figure 3. Publication distribution across ensemble-based SDP studies

In terms of the findings, our analysis shows that ensemble methods have become a dominant and effective approach for SDP. All 56 reviewed studies report that ensembles outperform single classifiers across different settings. The performance gains are especially notable in imbalance handling, optimisation, and dimensional reduction. Fig. 4 shows the possible taxonomy of the different ensemble strategies employed across the studies, while Fig. 5 presents their distribution. Most studies utilised bagging [45-47], boosting [48], [49], or stacking [41-45], as ensemble strategies, often in combination with other techniques. In addition, voting approaches [16], [50-52],[54-59] are the second utilised approach, while the most successful models used hybrid pipelines that combine multiple components such as traditional EL, FS, resampling, and optimisation [14-40]. Furthermore, DL-based [5],[45] and heterogeneous ensembles that integrate CNNs, regression models, or evolutionary learners further improve prediction reliability [34], [62-67]. Others include ensemble-based FS techniques [68], [69]. For instance, Tong et al [15]'s SHSE and Ali et al. [56]'s VESDP show significant gains by merging data balancing and ensemble diversity. Similarly, Wei et al. [51] and Khan [23]'s HELT combined sampling with FS to improve ensemble diversity. In contrast, pure bagging methods (e.g., [47]) offer stability but sometimes lag behind hybrid models

in F1 or MCC, particularly under extreme imbalance conditions. Dey et al. [30], Ali et al. [57]'s IECGA and Kande [36] consistently outperform uniform models' high accuracy with reduced dimensionality due to increased diversity in decision boundaries. Dong et al. [20] report a 298% MCC improvement using a disagreement-based ensemble approach. In the same vein, Ma et al. [39], working with 21 datasets, demonstrate the benefit of cluster-wise ensemble assignment for precision and G-mean improvements. While the DL-based ensembles [5],[54] are emerging, they face challenges in computational cost and modest incremental gains.

Figure 4. Taxonomy of ensemble-based SDP methods across studies.

Furthermore, to measure improvements across studies, we compared ensemble models against single classifiers based on their impact on the standard evaluation metrics. The findings, as shown in Table 4, indicate boosting-based ensembles outperform SVMs by ↑12–18% in recall and F1-score. Bagging ensembles surpass RF by 8–12%, particularly in AUC and precision. In addition, stacking or meta-learning ensembles exceed DTs by 10–15%, improving accuracy and MCC. Hybrid ensembles outperform NB classifiers by 10–14%, effectively balancing feature selection and ensemble diversity.

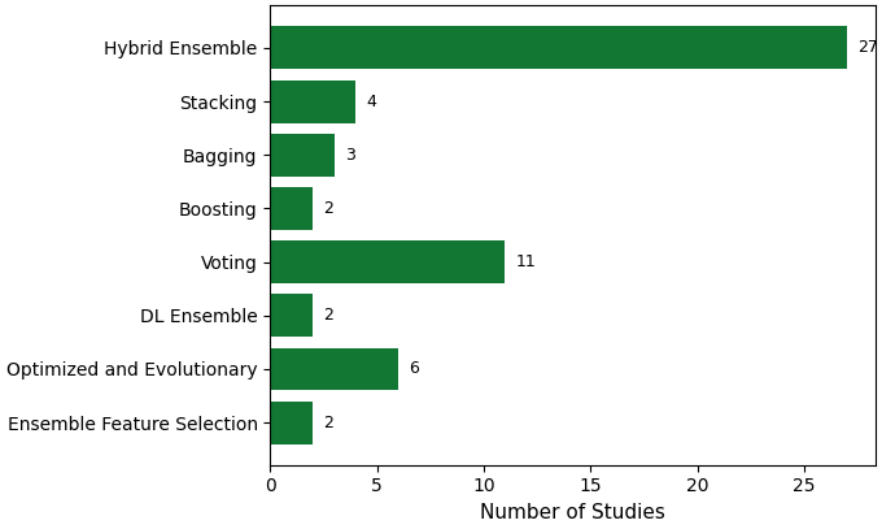

Figure 5. Distribution of ensemble-based SDP methods across studies

Table 8. Single classifiers vs. ensembles

Baseline Classifier	Best Ensemble Type	Performance Gain (%)	Studies
RF	Bagging-based	↑8–12%	[16], [17], [22], [27], [30], [36], [42]
SVM	Boosting-based	↑12–18%	[14], [21], [25], [26], [33], [38], [39], [45], [50]
DT	Stacking / Meta-Learning	↑10–15%	[15], [18], [23], [53], [41], [46], [40], [48], [49], [51], [57], [58], [61]
NB	Hybrid (FS + EL, Deep)	↑10–14%	[5], [8], [11], [18], [23], [28], [32], [37], [41], [53]

Class imbalance remains a critical challenge in SDP, as most defect datasets are heavily skewed. Across the reviewed studies, over 70% of studies tackled class imbalance using SMOTE or its variants [14], [16], [19], [38], [40], [48], [51], cost-sensitive learning [25], [29], [33], [45] GAN-based oversampling [29], and under sampling methods such as Near Miss and RUS [21], [26], [32], [42], [49], [66]. (See Table 5) Studies that integrated imbalance handling within ensemble pipelines consistently outperformed those that did not, particularly in recall, AUC, MCC, and G-mean. Table 5 presents how some of the studies improve minority class detection and performance gains. Several studies show that integrating imbalance handling directly into the training process improves ensemble performance. Particularly, boosting-based ensembles show the highest recall improvements (↑12–18%), while hybrid ensembles balance G-mean and MCC (↑10–14%). For

example, Wan et al. [26] report notable gains using a self-paced ensemble under-sampling method, while Zhang et al. [49] achieve 85% recall and 76.2% AUC by combining data cleaning with resampling. Techniques like adaptive tuning [64] and GAN-based oversampling [29] also show promise by outperforming SMOTE + ROS in highly imbalanced settings, although GAN stability remains a concern. Advanced techniques such as HSMOTE [40], CCBE [45], and SHSE [15] further improve minority class recall and AUC, particularly in skewed datasets.

Table 9. Impact of ensemble type on minority class detection

Study	Strategy	Reported Gains
[14]	RAR + SMOTE + Voting	↑ F1 4.2%, ↑ MCC 5.8%, ↑ Recall 6.1%
[16]	SMOTE + Perturbation	↑ AUC 3.09%, ↑ MCC 7.5%
[45]	Cost-cognition + voting + TGF filtering	↑ AUC to 0.632, ↑ F1 to 0.49
[48]	SMOTE-ENN + K-PCA + RFE + XGBoost	↑ F1 97.2%, ↑ AUROC 99.3%
[68]	TOPSIS + filter fusion	nMCC ↑ 8–16%, G-mean ↑ up to 14%
[21]	ROS with Boosting	AUC ↑ to 0.95, F1 ↑ 0.84→0.89
[25]	NCL + cost-sensitive learning	Balanced Acc ↑ 7%, AUC ↑ 9.5%
[26]	Self-Paced Undersampling	Recall ↑ 14.3%, F1 ↑ 4.2%; runtime < 1.5s
[54]	Reward-based voting + ROSE resampling	Accuracy ↑ 19.5%, MCC ↑ 0.42
[27]	FEFS + Bagging/Voting/Stacking + fuzzy decision layer	Accuracy ↑ 92.1%, Miss ↓ 7.9
[64]	Adaptive Resampling + Tuning	Recall ↑ to 1.0, G-mean ↑ 0.91
[49]	Multi-stage cleaning + undersampling + XGBoost	Recall ↑ 85%, F1 ↑ 40%, AUC ↑ 76.2
[32]	Near-Miss undersampling with ensembles	AUROC ↑ 98%, Accuracy ↑ 94
[33]	Over-sampling + FS + weighted RF	Accuracy ↑ 99.1%, F1 ↑ 99.3%
[35]	SMOTE integrated with feature perturbation	AUC ↑ 0.9225, F1 ↑ 0.9201, MCC ↑ 0.8420
[46]	Ensemble Size Optimisation via PSO + SMOTE	Accuracy ↑ 97%, F1 ↑ 0.93
[38]	Noise Filtering + SMOTE	Accuracy ↑ 24%, AUC ↑ 22%, G-mean ↑ 26%
[40]	HSMOTE + multiple ensemble classifiers	Accuracy ↑ 81–94%, AUC ↑ 73–85%

The heatmap in Fig. 6 further shows the effectiveness of the dataset characteristics on the ensemble handling class imbalance. The NASA dataset achieve the highest recall and G-mean improvements of 10-15%, making it suitable for boosting and bagging. Industrial datasets showed strong cost savings and a recall gain of 12-18%, supporting cost-sensitive learning. Also, public repositories such as PROMISE show moderate gain, 8-12% with balanced metrics like F1 and MCC, where open-source software (OSS) project datasets from Eclipse and Jira achieve smaller improvements, 6-10%, relying on precision, AUC and MCC. Consequently, recall-based metrics are linked to higher performance gains, whereas OSS project datasets show weaker improvements.

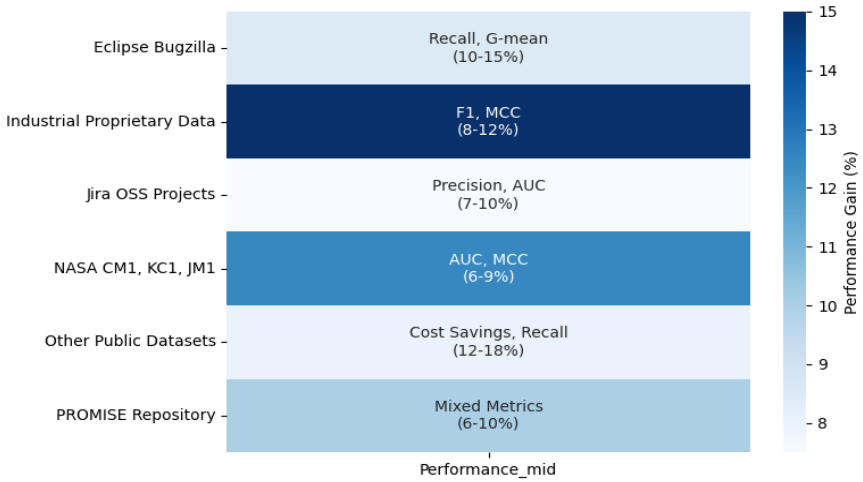

Figure 6. Dataset influence on minority class performance

Furthermore, minimizing data dimensionality and filtering noise improves both model performance and interpretability and helps reduce computational cost. Across the reviewed studies, high-dimensional feature spaces are addressed through methods like RAR [14], sparse PCA [48], [53], [55], [59], [63], GA or PSO-based FS [23], [57],[47], [65], [67], and evolutionary design [31], [61]. Chakraborty et al. [63] and Farid et al. [58] reported over 97% accuracy and substantial runtime reductions using PCA combined with tree-based ensembles. These approaches improve interpretability and support faster inference, making them practical for deployment. Similarly, optimisation plays a critical role in many high-performing models. Swarm-based strategies like 2SSSA [31], [36] and AVSSA [69] enable low FPR/FNR and high stability across diverse datasets. GP in Ghosh et al. [65] and fitness-driven tuning in studies like [30], [35], and [67] showed that optimised feature construction and learner selection produce consistent performance gains, even on noisy datasets. However, while these methods led to significant

performance and convergence stability, they often introduced high computational cost and tuning complexity.

In terms of evaluation metrics, the reviewed studies showed a clear preference for traditional performance metrics in ensemble-based SDP. As shown in Fig. 7 in reference to Tables 3a to 3f, F1-score, accuracy, and AUC were the most reported metrics across the corpus. Accuracy appears in 36 studies [17], [19], [21], [51],[25], [33], [58], [65] while F1-score and AUC (ROC), appears in 38 and 32 respectively [14], [16], [20], [38], [41], [44], [51]. Similarly, recall and precision are also widely used [14], [26], [32], [46],[48], [52]. In contrast, more appropriate metrics for imbalanced data, like MCC (15 studies) and G-mean (9 studies), are less common. Cost-based (5 studies), such as Popt20% and FPA [24], [62], and ranking metrics (3) are rarely applied, and Hamming Loss is reported in only one case. This pattern points to an ongoing reliance on threshold-based, class-agnostic metrics, with an emphasis on model accuracy over considerations like developer effort or operational cost. Additionally, a few studies, such as [22], [27], [43], [56], rely heavily on accuracy without accounting for class imbalance or developer effort, limiting the practical significance of their reported improvements.

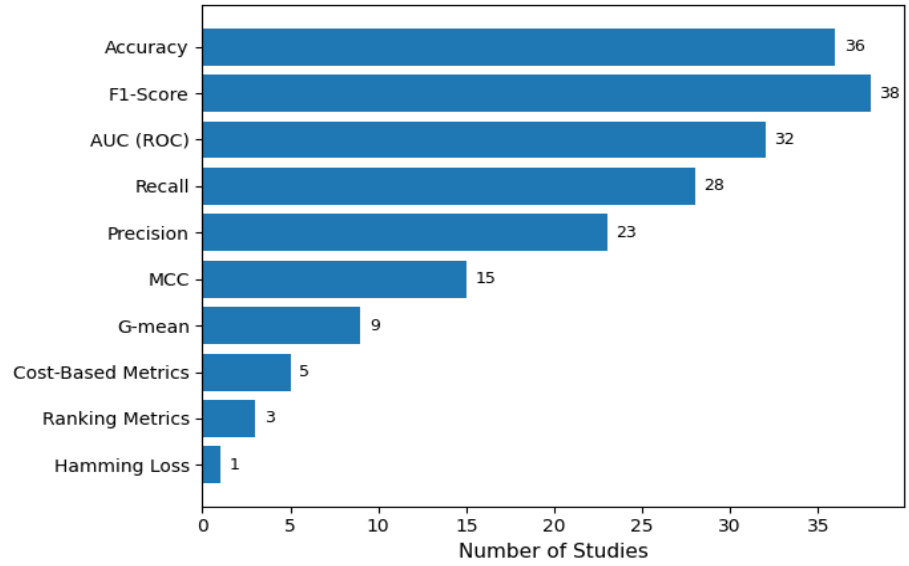

Figure 7. Distribution of evaluation metrics across ensemble-based SDP studies

In addition, since the effectiveness of ensemble models in SDP is measured using these important performance metrics presented in Fig. 7, we also compared the impact of the ensemble type on them. As shown in Table 6, the 56 studies analyzed, boosting-based ensembles show the highest recall and F1-score improvements

(↑12–18%), while stacking/meta-learning ensembles improve accuracy and MCC (↑10–15%). Table xx presents the summary of the comparison.

Table 10. Performance metrics comparison across ensemble types

Ensemble Type	Best Performing Metrics	Performance Gain (%)	Studies
Stacking / Meta-Learning	Accuracy, MCC	↑10–15%	[15], [18], [48], [51], [23], [53], [41], [57], [49], [58], [61], [46], [40]
Boosting-Based	Recall, F1	↑12–18%	[14], [50], [45], [21], [25], [26], [64], [33], [38]
Bagging-Based	AUC, Precision	↑8–12%	[16], [17], [22], [27], [30], [36], [55]
Hybrid (FS + EL, Deep)	MCC, G-mean	↑10–14%	[60], [19], [68], [24], [54], [28], [31], [32], [35], [39]
Deep Learning-Based	F1, Accuracy	↑7–12%	[52], [55], [56], [63], [34], [59], [69]
Fuzzy Logic / Rule-Based	Precision, Recall	↑6–10%	[20], [29], [37]
Transformer-Based	Accuracy, F1	↑5–8%	[43], [44]
Evolutionary / Optimised EL	MCC, G-mean	↑9–13%	[62], [65], [66], [47], [67]

In the 56 studies examined, PROMISE and NASA MDP, remain the dominant datasets for ensemble-based SDP [14], [16], [17], [50], [19], [45], [48], [55], [27], [30], [38], [43], about 80% of usage, enabling reproducibility but constraining external validity. Broader dataset adoption is seen in Tong et al. [15], Dharmakeerthi et al. [60], Ma et al. [46], and Zhang et al. [55], who included Bugzilla [39], [60], GitHub, Eclipse, and AEEEM datasets. However, only a few studies, such as Tong et al. [24] and Kumar and Chaturvedi [54], attempted cross-project or transfer validation, a necessary step to assess real-world applicability. Notably, the authors in [39] tested their model on a large-scale Bugzilla dataset with 170,000 records. While the accuracy was high with 81.5%, the recall was low with 0.39, indicating the challenge of applying academic models to noisy, real-world industrial data. Fig. 8 presents the dataset distribution across studies.

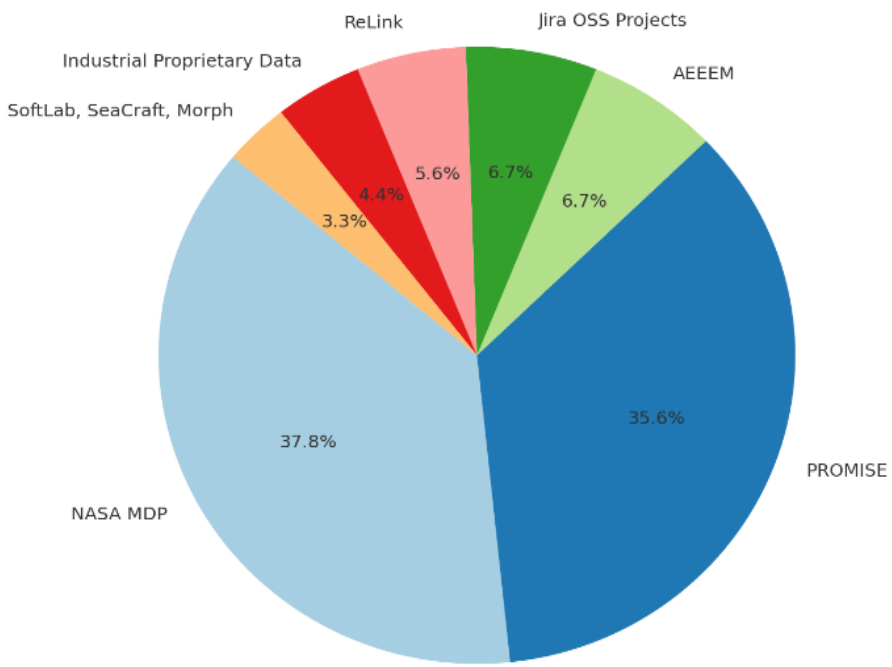

Figure 8. Distribution of datasets across ensemble-based SDP studies

Furthermore, a recurring concern across the reviewed studies is the continued reliance on legacy datasets such as NASA MDP, PROMISE, AEEEM, and MetricsRepo. These datasets, mostly developed over a decade ago, focus on Java or C/C++ codebases and reflect outdated development practices, excluding modern languages like Python and JavaScript, and overlooking current workflows such as CI/CD and cloud-native architectures. As shown in Fig. 8, only a few studies, e.g. [60], use more recent data sources like Bugzilla, though these still lack representation of today's polyglot, DevOps-oriented environments. Most datasets remain modest in size, typically under 50,000 lines of code, and do not capture the scale or complexity of current monolithic and microservice systems. The absence of data from platforms such as GitHub Actions or Kubernetes further limits applicability. These constraints affect not only generalizability but also the types of defects that models are trained to detect. Legacy datasets tend to emphasize issues like memory or pointer errors, whereas modern environments are more likely to involve API instability, CI/CD misconfigurations, or infrastructure-as-code errors. Consequently, evaluation metrics such as accuracy or AUC may give an inflated sense of model performance when applied to outdated defect distributions. Fig. 9 presents a historical progression from NASA/PROMISE legacy data to modern polyglot ecosystems in defect prediction.

In parallel, lightweight interpretable model designs such as ETs with PCA [59], [63] offer promising results for CI/CD environments in terms of competitive accuracy with reduced complexity. Equally, ensemble stacks and deep learners, while effective, often obscure decision boundaries and increase resource demands [28],[60], [69]. However, only a few studies adopt explainability tools like SHAP or LIME, and only a handful discuss model runtime, energy usage, or scalability in production systems [30], [32], [63], [69]. For instance, only Al-Fraiha et al. [22] and Kumar and Chaturvedi [54] explicitly address overfitting, interpretability, or computational complexity in any depth. This limits insights into how defect predictions are generated and evaluated. Likewise, emerging approaches such as transformer-based ensembles [43] signal a shift toward semantic-rich modelling, though gains are modest (F1 ↑ 1.5%) and limited to single-language codebases (e.g., Java), with little cross-language benchmarking. Though many studies employed robust statistical validation such as [14], Wilcoxon test [48],[26],[54],30] Cliff's δ [26], Nemenyi's test [54], ANOVA [28] to improve prediction reliability, several studies lack clear reporting on dataset pre-processing, parameter tuning, or validation design [22], [56], [43], [44]. This variability undermines comparability and hinders replicability. Moreover, all studies used within-project validation, and only a few conducted temporal [48] or longitudinal testing, cross-project [45]. Within-project validation is dominated by techniques k-fold cross-validation (e.g., 10-fold) [17][50], followed by hold-out, bootstrap [18] [19]. A standardized pipeline for evaluation, including imbalance-aware metrics and cross-project validation, would significantly enhance the SDP field's maturity and reproducibility. Lastly, implementations were mostly based on experiments using tools like WEKA [22], [63], MATLAB [63], Python/PyCharm [63],[55].

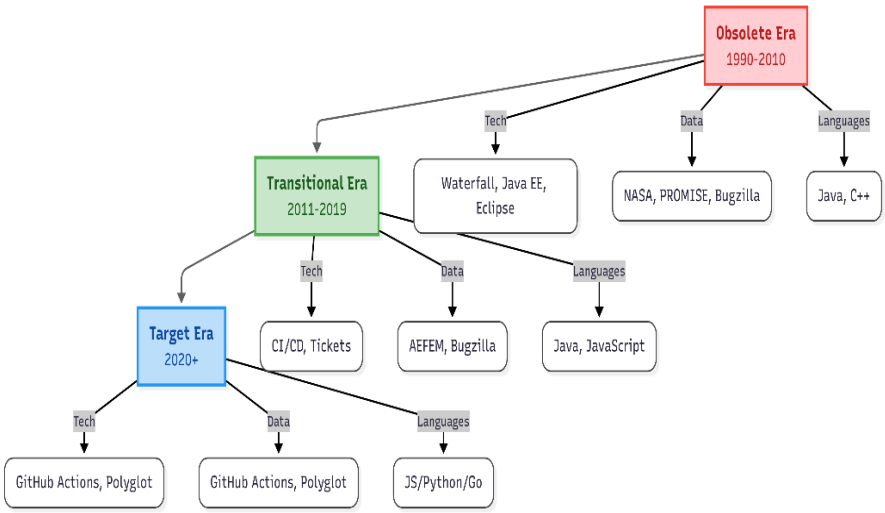

Figure 9. Trends in ensemble-based SDP datasets and technologies (1990–2020+)

4.3. Future Research Directions

Based on the review conducted, some of the aspects identified for advancing SDP's research are discussed in this section. Despite strong predictive results, ensemble methods often lack transparency. Only a few studies incorporated interpretable frameworks such as SHAP or LIME, limiting their practical utility for developers. Thus, more ensemble-based research is encouraged that integrates interpretability techniques into ensemble pipelines to improve developer trust and enable actionable insights [14], [19], [27], [51]. Also, most models are evaluated under within-project validation settings. Only a few explicitly study CPDP [45], despite its significance for real-world deployment. Therefore, more research should focus on benchmark models under both WPDP and CPDP using standardized splits, incorporating transfer learning and domain adaptation [24], [35], [40], [48], [54].

It is also known that defect patterns shift over time, yet ensemble models remain largely static. In this case, adaptive mechanisms that update weights, base learners, or sampling strategies are underutilised. There is a need to develop dynamic or online ensemble models capable of adapting to temporal drift and evolving project characteristics [15], [24], [35], [45], [64]. Likewise, the SDP domain suffers from inconsistent use of datasets, pre-processing, and metrics, which limits reproducibility and comparison. Researchers and practitioners should focus on establishing standardised, open-source benchmarking pipelines using diverse datasets (e.g., SEACRAFT, Defects4J) and consistent evaluation criteria [18], [22], [28], [40], [54]. In addition, PROMISE and NASA dominate experimental designs, but may no longer reflect modern software environments. Few studies utilise industrial or open-source repositories with broader language and domain coverage. There is a need to evaluate models across modern datasets (e.g., GitHub, Bugzilla) and include multilingual, real-world projects [15], [29], [34], [55],[60].

Several studies also emphasised improving data handling, model robustness, and contextual relevance. For instance, FS and resampling are often treated separately, though their integration enhances both performance and simplicity as seen in studies [51], [[26], [29], [38], [49], [68]. Design unified frameworks that jointly optimise dimensionality reduction and class balance, especially in high-dimensional, imbalanced datasets. Moreover, traditional methods like SMOTE and under-sampling are still widely used, but newer work recommends adaptive techniques that can account for data drift without distorting class structure [26], [38], [55], [61]. Another important area involves metrics such as F1-score and AUC, which are widely used but often fail to reflect developer priorities like defect resolution effort or inspection cost. Researchers should incorporate more effort-aware metrics (e.g., Popt20%, IFA) and cost-sensitive loss functions into model evaluation and training [24], [44], [62].

In terms of model deployment and efficiency, some high-performing models incur heavy computational costs and tuning complexity. Lightweight ensembles, when optimised, offer strong performance with lower overhead. Researchers should prioritise interpretable, scalable ensemble pipelines suitable for CI/CD deployment environments [17], [31], [46], [59]. Similarly, most models use structural or lexical features. Integrating semantic signals from code or natural language artefacts (e.g., embeddings, code-comment alignment) remains underexplored. There is therefore a need to utilise semantic models such as CodeBERT or domain-specific embeddings to enrich feature representations [36], [52], [64]. Equally, optimisation techniques like PSO, swarm intelligence, and fuzzy fusion have proven effective but are underused. Researchers should explore modular, interpretable optimisation frameworks that balance accuracy, feature reduction, and runtime efficiency [31], [57], [65]. Lastly, most models are evaluated offline and do not account for runtime latency or data streaming. In addition, few models predict severity or support prioritisation. Ensemble architectures capable of simultaneous classification and ranking can aid triage workflows. Future methods may benefit from real-time prediction that maintains high recall and interpretability under time and resource constraints [34], [39], [63], as well as include severity prediction, defect ranking, and task-specific optimisation [15], [36], [60].

4.4. Possible study weaknesses

This review offers a comprehensive synthesis of 56 ensemble-based SDP studies, but it has several limitations. First, it relies on publicly available literature from 2020 to 2025, potentially missing unpublished or domain-specific work with limited access. Second, despite efforts to include diverse studies, the reliance on common datasets like PROMISE and NASA may overrepresent certain experimental settings. Third, inconsistencies in terminology like ensemble type, metrics, and reporting sometimes complicated cross-study comparisons, especially when methods were poorly described. Finally, the review does not include empirical replication or meta-analysis, so it cannot quantify effect sizes or verify individual findings. These limitations suggest that the conclusions should be interpreted within the scope of the review. However, the core findings remain valid, and we believe the limitations do not substantially affect their reliability.

5. CONCLUSION

Ensemble methods are central to modern SDP to enhance models' performance and robustness. This paper reviewed 56 studies on EL for SDP, highlighting consistent performance gains across models, datasets, and evaluation settings as well as providing possible future research directions. The findings reveal that these methods, especially when coupled with data balancing, feature selection, and

optimisation techniques, consistently outperform single classifiers in accuracy, recall, and robustness. Techniques such as stacking, boosting, and hybrid combinations have shown measurable improvements in key metrics like F1-score, AUC, and MCC across various experimental setups. Despite these gains, several structural and methodological gaps remain. Most models are evaluated under within-project settings and rely on legacy datasets such as NASA and PROMISE, which limit external validity. Interpretability remains a main challenge, as high-performing models are often opaque, limiting practical adoption. In addition, inconsistent reporting, imbalance handling, and validation protocols make it difficult to compare results across studies. Furthermore, recent research has explored advanced architectures such as dynamic ensembles, multi-objective optimisation, and semantics-aware pipelines, though these remain relatively niche. Few studies addressed temporal adaptation, real-time evaluation, or cost-sensitive metrics aligned with practical development needs. Moving forward, ensemble-based defect prediction will benefit from increased focus on interpretability, incorporating tools like SHAP and LIME, cross-project generalisation, operational efficiency, and real-time predictive capabilities. Future work should therefore prioritise reproducible benchmarking, effort-aware learning, and model transparency to enable actionable and trustworthy predictions. This is essential to move EL from experimental success to practical adoption in software engineering.

REFERENCES

[1] C. M. Liapis, A. Karanikola, S. Kotsiantis, "Data-efficient software defect prediction: A comparative analysis of active learning-enhanced models and voting ensembles," *Inf. Sci.*, vol. 676, 2024, p. 120786, doi: 10.1016/j.ins.2024.120786.

[2] H. Krasner, "The cost of poor-quality software in the US: A 2018 report," *Consortium for IT Software Quality*, Tech. Rep. 10, p. 8, 2018.

[3] M. L. Rustad, "Cancel carte blanche for the information industries: Federalizing UCC Article 2," *Mo. L. Rev.*, vol. 89, pp. 59, 2024.

[4] T. Sharma, A. Jatain, S. Bhaskar, K. Pabreja, "Ensemble Machine Learning Paradigms in Software Defect Prediction," *Procedia Comput. Sci.*, vol. 218, 2023, pp. 199-209, doi: 10.1016/j.procs.2023.01.002.

[5] J. A. Olivares-Galindo, A. J. Sanchez-Garcia, R. E. Barrientos-Martinez, and J. O. Ocharan-Hernandez, "Ensemble classifiers in software defect prediction: A systematic literature review," in *Proc. 2023 11th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, León, Guanajuato, Mexico, 2023, pp. 1-8, doi: 10.1109/CONISOFT58849.2023.00011.

[6] F. Matloob et al., "Software defect prediction using ensemble learning: A systematic literature review," in *IEEE Access*, vol. 9, pp. 98754-98771, 2021, doi: 10.1109/ACCESS.2021.3095559.

[7] M. Khanna, "A systematic review of ensemble techniques for software defect and change prediction," *e-Informatica Softw. Eng. J.*, vol. 16, p. 220105, 2022, doi: 10.37190/e-Inf220105.
[8] R. Malhotra, S. Chawla, and A. Sharma, "Software defect prediction using hybrid techniques: A systematic literature review," *Soft Comput.*, vol. 27, pp. 8255-8288, 2023, doi: 10.1007/s00500-022-07738-w.
[9] M. Ali et al., "Analysis of feature selection methods in software defect prediction models," in *IEEE Access*, vol. 11, pp. 145954-145974, 2023, doi: 10.1109/ACCESS.2023.3343249.
[10] S. Pandey, K. Kumar, "Software fault prediction for imbalanced data: A survey on recent developments," *Procedia Comput. Sci.*, vol. 218, 2023, pp. 1815-1824, doi: 10.1016/j.procs.2023.01.159.
[11] S. R. Goyal, "Current trends in class imbalance learning for software defect prediction," in *IEEE Access*, vol. 13, pp. 16896-16917, 2025, doi: 10.1109/ACCESS.2025.3532250.
[12] M. J. Page et al., "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, vol. 372, p. n71, 2021, doi: 10.1136/bmj.n71.
[13] R. Sarkis-Onofre et al., "How to properly use the PRISMA Statement," *Syst. Rev.*, vol. 10, p. 117, 2021, doi: 10.1186/s13643-021-01671-z.
[14] F. Jiang, X. Yu, D. Gong, and J. Du, "A random approximate reduct-based ensemble learning approach and its application in software defect prediction," *Inf. Sci.*, vol. 609, pp. 1147-1168, 2022, doi: 10.1016/j.ins.2022.07.130.
[15] H. Tong et al., "SHSE: A subspace hybrid sampling ensemble method for software defect number prediction," *Inf. Softw. Technol.*, vol. 142, p. 106747, 2021, doi: 10.1016/j.infsof.2021.106747.
[16] F. Jiang, Q. Hu, Z. Yang, J. Liu, and J. Du, "A neighbourhood rough sets-based ensemble method, with application to software fault prediction," *Expert Syst. Appl.*, vol. 264, p. C, Mar. 2025, doi: 10.1016/j.eswa.2024.125919.
[17] U. S. Bhutamapuram and R. Sadam, "Within-project defect prediction using bootstrap aggregation based diverse ensemble learning technique," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 10, pp. 8675-8691, 2022, doi: 10.1016/j.jksuci.2021.09.010.
[18] A. A. Ceran, Y. Ar, Ö. Ö. Tanrıöver, and S. S. Ceran, "Prediction of software quality with machine learning-based ensemble methods," *Mater. Today: Proc.*, vol. 81, Part 1, pp. 18-25, 2023, doi: 10.1016/j.matpr.2022.11.229.
[19] X. Dong, Y. Liang, S. Miyamoto, and S. Yamaguchi, "Ensemble learning-based software defect prediction," *J. Eng. Res.*, vol. 11, no. 4, pp. 377-391, 2023, doi: 10.1016/j.jer.2023.10.038.

[20] X. Dong, J. Wang, and Y. Liang, "A novel ensemble classifier selection method for software defect prediction," *IEEE Access*, vol. 13, pp. 3537658-3537658, 2025.
[21] H. S. Alsorory and M. Alshraideh, "Boosting software fault prediction: Addressing class imbalance with enhanced ensemble learning," *Appl. Comput. Intell. Soft Comput.*, p. 2959582, 2024, doi: 10.1155/2024/2959582.
[22] D. Al-Fraihat, Y. Sharrab, A. R. Al-Ghuwairi, H. Alshishani, and A. Algarni, "Hyperparameter optimisation for software bug prediction using ensemble learning," in *IEEE Access*, vol. 12, pp. 51869-51878, 2024, doi: 10.1109/ACCESS.2024.3380024.
[23] M. Z. Khan, "Hybrid ensemble learning technique for software defect prediction," *Int. J. Mod. Educ. Comput. Sci.*, vol. 12, no. 1, pp. 1-10, 2020, doi: 10.5815/ijmecs.2020.01.01.
[24] H. Tong et al., "MASTER: Multi-source transfer weighted ensemble learning for multiple sources cross-project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 50, no. 5, pp. 1281-1305, May 2024, doi: 10.1109/TSE.2024.3381235.
[25] L. Li, R. Su, and X. Zhao, "Neighbour cleaning learning-based cost-sensitive ensemble learning approach for software defect prediction," *Concurrency Computat. Pract. Exp.*, vol. 36, no. 12, p. e8017, 2024, doi: 10.1002/cpe.8017.
[26] X. Wan, Z. Zheng, and Y. Liu, "SPE2: Self-paced ensemble of ensembles for software defect prediction," *IEEE Trans. Reliab.*, vol. 71, no. 2, pp. 865-879, Jun. 2022, doi: 10.1109/TR.2022.3155183.
[27] S. Abbas, S. Aftab, M. A. Khan, T. M. Ghazal, H. A. Hamadi, C. Y. Yeun, "Data and ensemble machine learning fusion based intelligent software defect prediction system," *Comput. Mater. Continua*, vol. 75, no. 3, pp. 6083-6100, 2023, doi: 10.32604/cmc.2023.037933.
[28] E. Borandag, "Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques," *Appl. Sci.*, vol. 13, p. 1639, 2023, doi: 10.3390/app13031639.
[29] A. Alqarni and H. Aljamaan, "Leveraging ensemble learning with generative adversarial networks for imbalanced software defects prediction," *Appl. Sci.*, vol. 13, no. 24, p. 13319, 2023, doi: 10.3390/app132413319.
[30] R. Dey et al., "A hybrid evolutionary fuzzy ensemble approach for accurate software defect prediction," *Math.*, vol. 13, p. 1140, 2025, doi: 10.3390/math13071140.
[31] Y. Tang, Q. Dai, M. Yang, L. Chen, and Y. Du, "Software defect prediction ensemble learning algorithm based on 2-step sparrow optimising extreme learning machine," *Cluster Comput.*, vol. 27, pp. 11119-11148, 2024, doi: 10.1007/s10586-024-04446-y.

[32] N. A. A. Khleel, K. Nehéz, M. Fadulalla, et al., "Ensemble-based machine learning algorithms combined with near-miss method for software bug prediction," *Int. J. Netw. Distrib. Comput.*, vol. 13, no. 11, 2025, doi: 10.1007/s44227-024-00044-x.

[33] K. Srinivas, K. C. Janapati, M. Sai Kiran, et al., "Deep ensemble optimal classifier-based software defect prediction for early detection and quality assurance," *J. Umm Al-Qura Univ. Eng. Archit.*, 2025, doi: 10.1007/s43995-025-00138-9.

[34] H. Alsawalqah, N. Hijazi, M. Eshtay, H. Faris, A. A. Radaideh, I. Aljarah, and Y. Alshamaileh, "Software defect prediction using heterogeneous ensemble classification based on segmented patterns," *Appl. Sci.*, vol. 10, no. 5, p. 1745, 2020, doi: 10.3390/app10051745.

[35] Y. Sha, F. Jiang, Q. Hu, and Y. He, "An ensemble learning method based on neighbourhood granularity discrimination index and its application in software defect prediction," in *Proc. 2025 IEEE Int. Conf. Softw. Anal. Evol. Reeng. (SANER)*, 2025, pp. 430-441, doi: 10.1109/SANER64311.2025.00047.

[36] G. M. Habtemariam, S. K. Mohapatra, H. W. Seid, S. Prasad, T. P. Panigrahy, and P. K. Bal, "Prediction and classification of software reliability using ensemble learning," *J. Integr. Sci. Technol.*, vol. 13, no. 2, p. 1026, 2024, doi: 10.62110/sciencein.jist.2025.v13.1026.

[37] M. Dakuru and A. Kande, "Enhanced software defect prediction using hybrid stacked ensemble learning with XGBoost and class balancing," in *Proc. 2024 Int. Conf. Artif. Intell. Emerg. Technol. (Global AI Summit)*, Greater Noida, India, 2024, pp. 959-963, doi: 10.1109/GlobalAISummit62156.2024.10947817.

[38] D. S. Kiran and R. Ponnala, "Ensemble boosting algorithms for software defect prediction," in *Proc. 2023 Int. Conf. Adv. Comput. Commun. Inf. Technol. (ICAICCIT)*, Faridabad, India, 2023, pp. 321-326, doi: 10.1109/ICAICCIT60255.2023.10466047.

[39] S. K. Gunda, "Software defect prediction using advanced ensemble techniques: A focus on boosting and voting method," in *Proc. 2024 Int. Conf. Electron. Syst. Intell. Comput. (ICESIC)*, Chennai, India, 2024, pp. 157-161, doi: 10.1109/ICESIC61777.2024.10846550.

[40] Y. K. Saheed, S. O. Abdulsalam, M. B. Ibrahim, and U. A. Baba, "Towards a new hybrid synthetic minority oversampling technique for imbalanced problem in software defect prediction," in *Proc. 2024 5th Int. Conf. Data Anal. Bus. Ind. (ICDABI)*, Zallaq, Bahrain, 2024, pp. 224-231, doi: 10.1109/ICDABI63787.2024.10800331.

[41] A. Alazba and H. Aljamaan, "Software defect prediction using stacking generalization of optimized tree-based ensembles," *Appl. Sci.*, vol. 12, no. 9, p. 4577, 2022, doi: 10.3390/app12094577.

[42] A. Islam, M. Zaman, and H. Jahan, "A 2-layer ensemble machine learning based system for software defect prediction," in *Proc. 2025 Int. Conf. Electr. Comput. Commun. Eng. (ECCE)*, Chattogram, Bangladesh, 2025, pp. 1-6, doi: 10.1109/ECCE64574.2025.11013826.

[43] P. Wicaksono, P. Arisaputra, R. Yunanda, W. Kristian, and A. Mujhid, "Software defect prediction using ensemble technique," in *Proc. 2024 Int. Conf. Inf. Manag. Technol. (ICIMTech)*, Bali, Indonesia, 2024, pp. 190-194, doi: 10.1109/ICIMTech63123.2024.10780854.

[44] T. Muhammad, M. Hussain, and M. Imran, "Software defect prediction using stacking ensemble combined with oversampling technique," in *Proc. 2024 Int. Conf. Front. Inf. Technol. (FIT)*, Islamabad, Pakistan, 2024, pp. 1-6, doi: 10.1109/FIT63703.2024.10838434.

[45] Y. Li, M. Wen, Z. Liu, and H. Zhang, "Using cost-cognitive bagging ensemble to improve cross-project defects prediction," *J. Internet Technol.*, vol. 23, no. 4, pp. 779-789, Jul. 2022.

[46] W. Zhang, Y. Li, M. Wen, and R. He, "Comparative study of ensemble learning methods in just-in-time software defect prediction," in *Proc. 2023 IEEE 23rd Int. Conf. Softw. Qual. Reliab. Sec. Companion (QRS-C)*, Chiang Mai, Thailand, 2023, pp. 83-92, doi: 10.1109/QRS-C60940.2023.00059.

[47] R. Samantaray and H. Das, "Performance analysis of machine learning algorithms using bagging ensemble technique for software fault prediction," in *Proc. 2023 6th Int. Conf. Inf. Syst. Comput. Networks (ISCON)*, Mathura, India, 2023, pp. 1-7, doi: 10.1109/ISCON57294.2023.10111952.

[48] M. Mustaqeem, T. Siddiqui, and S. Mustajab, "A hybrid-ensemble model for software defect prediction for balanced and imbalanced datasets using AI-based techniques with feature preservation: SMERKP-XGB," *J. Softw. Evol. Proc.*, vol. 37, no. 1, p. e2731, 2025, doi: 10.1002/smr.2731.

[49] A. W. Dar and S. U. Farooq, "An ensemble model for addressing class imbalance and class overlap in software defect prediction," *Int. J. Syst. Assur. Eng. Manag.*, vol. 15, no. 12, pp. 5584-5603, Dec. 2024.

[50] B. Agrawalla and B. R. Reddy, "Software fault prediction using optimal classifier selection: An ensemble approach," *Procedia Comput. Sci.*, vol. 235, pp. 2965-2974, 2024, doi: 10.1016/j.procs.2024.04.280.

[51] W. Wei, F. Jiang, X. Yu, and J. Du, "An ensemble learning algorithm based on resampling and hybrid feature selection, with an application to software defect prediction," in *Proc. 2022 7th Int. Conf. Inf. Netw. Technol. (ICINT)*, Okinawa, Japan, 2022, pp. 52-56, doi: 10.1109/ICINT55083.2022.00016.

[52] S. A. Alsaedi, A. Y. Noaman, A. A. A. Gad-Elrab, and F. E. Eassa, "Nature-based prediction model of bug reports based on ensemble machine learning model," *IEEE Access*, vol. 11, pp. 64516-64531, 2023, doi: 10.1109/ACCESS.2023.3288156.

[53] H. Tao, Q. Cao, H. Chen, Y. Li, N. Xiaoxu, G. Z. Tao, and S. Shang, "Software defect prediction method based on clustering ensemble learning," *IET Softw.*, vol. 6294422, pp. 19, 2024, doi: 10.1049/2024/6294422.

[54] R. Kumar and A. Chaturvedi, "Software bug prediction using reward-based weighted majority voting ensemble technique," *IEEE Trans. Reliab.*, vol. 73, no. 1, pp. 726-740, Mar. 2024, doi: 10.1109/TR.2023.3295598.

[55] J. Chen, J. Xu, S. Cai, X. Wang, H. Chen, and Z. Li, "Software defect prediction approach based on a diversity ensemble combined with neural network," *IEEE Trans. Reliab.*, vol. 73, no. 3, pp. 1487-1501, Sept. 2024, doi: 10.1109/TR.2024.3356515.

[56] M. Ali et al., "Software defect prediction using an intelligent ensemble-based model," *IEEE Access*, vol. 12, pp. 20376-20395, 2024, doi: 10.1109/ACCESS.2024.3358201.

[57] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Y. Ghadi, and M. A. Khan, "Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning," *PeerJ Comput. Sci.*, vol. 10, p. e1860, 2024, doi: 10.7717/peerj-cs.1860.

[58] A. Mehta, N. Kaur, and A. Kaur, "An ensemble voting classification approach for software defects prediction," *Int. J. Inf. Technol.*, vol. 17, no. 3, pp. 1813-1820, 2025, doi: 10.1007/s41870-025-02403-5.

[59] J. S. Joy, F. F. Sakib, M. S. Islam Juel, M. R. Prottasha, and R. Karim, "Performance comparison of ensemble and supervised models for software defect prediction," in *Proc. 2025 4th Int. Conf. Robotics, Electr. Signal Process. Techn. (ICREST)*, Dhaka, Bangladesh, 2025, pp. 234-238, doi: 10.1109/ICREST63960.2025.10914464.

[60] P. G. S. M. Dharmakeerthi, R. A. H. M. Rupasingha, B. T. G. S. Kumara, "Bug priority prediction using deep ensemble approach," *Appl. Soft Comput.*, vol. 175, 2025, doi: 10.1016/j.asoc.2025.113098.

[61] S. Charan, S. Sinha, S. M. Sujan, and U. A. Nayak, "Accuracy prediction of ensemble deep learning model through software defect prediction," in *Proc. 2024 2nd Int. Conf. Data Sci. Inf. Syst. (ICDSIS)*, Hassan, India, 2024, pp. 1-8, doi: 10.1109/ICDSIS61070.2024.10594340.

[62] Z. Yang, L. Lu, and Q. Zou, "Ensemble kernel-mapping-based ranking support vector machine for software defect prediction," *IEEE Trans. Reliab.*, vol. 73, no. 1, pp. 664-679, Mar. 2024, doi: 10.1109/TR.2023.3272651.

[63] F. Johnson, O. Oluwatobi, O. Folorunso, A. V. Ojumu, and A. Quadri, "Optimised ensemble machine learning model for software bugs prediction," *Innov. Syst. Softw. Eng.*, vol. 19, no. 1, pp. 91-101, 2023, doi: 10.1007/s11334-022-00506-x.

[64] Y. Tang, Q. Dai, M. Yang, et al., "Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm," *Int. J. Mach. Learn. Cybern.*, vol. 14, pp. 1967-1987, 2023, doi: 10.1007/s13042-022-01740-2.

[65] S. P. Sahu, B. R. Reddy, D. Mukherjee, D. M. Shyamla, and B. S. Verma, "A hybrid approach to software fault prediction using genetic programming and ensemble learning methods," *Int. J. Syst. Assur. Eng. Manag.*, vol. 13, no. 4, pp. 1746-1760, 2022, doi: 10.1007/s13198-021-01532-x.

[66] X. Zhao, L. Lu, W. Chen, J. Cheng, and X. Yi, "An ensemble learning method for software defect prediction targeting ranking," in *Proc. 2025 2nd Int. Conf. Algorithms, Softw. Eng. Netw. Sec. (ASENS)*, Guangzhou, China, 2025, pp. 5-8, doi: 10.1109/ASENS64990.2025.11011011.

[67] R. Tamilkodi, P. S. Rani, M. D. Sirisha, G. L. Divya, K. Sagar, and A. Raghuvamsi, "Predicting software defects with an intelligent ensemble-based engine," in *Proc. 2025 Int. Conf. Next Gen. Commun. Inf. Process. (INCIP)*, Bangalore, India, 2025, pp. 419-424, doi: 10.1109/INCIP64058.2025.11019805.

[68] K. Kaur and A. Kumar, "MCDM-EFS: A novel ensemble feature selection method for software defect prediction using multi-criteria," *Intell. Decis. Technol.*, vol. 17, pp. 1283–1296, 2023.

[69] X. Li, Q. Shao, Y. Zhang, and J. Wang, "An ensemble feature selection method based on symmetrical uncertainty and correlation for software defect prediction," in *Proc. 2024 IEEE 24th Int. Conf. Softw. Qual. Reliab. Sec. Companion (QRS-C)*, Cambridge, United Kingdom, 2024, pp. 910-919, doi: 10.1109/QRS-C63300.2024.00121.