

Software Defect Prediction: A Systematic Review and a Conceptual Framework for Stable, Actionable Explanations (AISD-F)

Dikeledi Thubisi¹, Bassey Isong², Rodney Sebopelo³

^{1,3}Computer Science Department, North-West University, Potchefstroom, South Africa

²Computer Science Department, North-West University, Mafikeng, South Africa

Received:

June 6, 2026

Revised:

July 21, 2026

Accepted:

September 10, 2026

Published:

October 24, 2026

Corresponding Author:

Author*:

Dikeledi Irene Thubisi

Email*:

irenethubisi@gmail.com

DOI:

10.63158/journalisi.v8i5.1823

© 2026 Journal of Information Systems and Informatics. This open access article is distributed under a (CC-BY License)



Abstract. Machine learning-based software defect prediction (SDP) has traditionally focused on improving predictive accuracy; however, recent research increasingly emphasizes explanations that are understandable and actionable for developers. This systematic review examines the use of Explainable Artificial Intelligence (XAI) in SDP and evaluates evidence concerning explanation reliability in practice. Following PRISMA guidelines, 40 peer-reviewed studies, primarily published between 2020 and 2025, were selected from IEEE Xplore, SpringerLink, ScienceDirect, Google Scholar, and Scopus. The studies were analyzed according to XAI methods, explanation reliability and stability, dataset quality, and model selection. LIME and SHAP were the most commonly used methods. However, multi-run evaluations revealed changes in feature rankings across configurations, contrasting with the reliability reported in many single-run studies. Data-quality issues—particularly SZZ-related label noise, class imbalance, and multicollinearity—often affected prediction reliability more substantially than model architecture. The review also identified increasing use of line-level and just-in-time prediction, although industrial-scale validation remains limited. No reviewed study provided controlled evidence that XAI explanations improve developers' ability to resolve software defects. Accordingly, this paper proposes AISD-F, a conceptual five-stage modular framework integrating data-quality assurance, hybrid model selection, dual-layer explanation, and CI/CD as a research agenda rather than a deployment-ready solution.

Keywords: Software defect prediction; Explainable Artificial Intelligence; explanation stability; class imbalance; SLR

1. INTRODUCTION

In software quality assurance (SQA) and maintenance, software defect prediction (SDP) is one of several tasks performed in the software development life cycle (SDLC), focusing on detecting fault-prone software modules before the testing phase [1], [2], [3]. SDP makes it possible for development teams to allocate scarce testing resources efficiently by analysing software metrics and historical data. Resource allocation in software development is closely tied to early fault identification. Defect detection and correction can account for more than 50% of overall development costs [1], [4], making software defect prediction (SDP) an important practical concern. Earlier SDP approaches relied mainly on statistical methods, whereas recent work has increasingly adopted machine learning (ML) and deep learning (DL), motivated by their reported classification performance on software quality datasets [3], [4], [5]. Traditional models, including Decision Trees (DT) and Logistic Regression (LR), provide relatively direct interpretations of their predictions. However, the use of more complex DL architectures, such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, has increased the need for Explainable AI (XAI). The literature also uses several related terms that require distinction. Explainability describes the extent to which a model's internal logic can be expressed in human terms [6]. Interpretability refers to the extent to which a person can predict a model's output from its inputs without additional tools [6], [7]. Explanation stability concerns whether an explanation method produces consistent outputs for the same instance across repeated runs or configurations [8], [9]. Actionability indicates whether an explanation identifies a concrete step a developer can take, such as refactoring a specific method or reviewing a specific commit [8] - [10]. This review focuses on file-, commit-, and line-level SDP. A small number of closely related tasks: vulnerability localisation, security-issue classification, project-failure prediction, and testing-effort prediction, are included only where they share XAI methodology directly relevant to SDP. These are flagged as adjacent-task studies rather than core SDP studies throughout.

While achieving strong predictive performance, these models operate as black boxes; their reasoning is difficult for developers to understand [6], [7]. The need for interpretability arises because, regardless of their accuracy, it is difficult to establish a model's trustworthiness and apply it in real-world situations. An example of this is a

model that predicts whether a software module is defective or not without explanation, would provide less useful guidance for developers than the statement “this method contains many lines of code (LOC),” which encourages action. In particular, developers find it difficult to identify which code features contributed to the prediction and whether they could verify that the model is relying on meaningful information rather than fitting noise [4], [6], [8], [9], [10]. The absence of such transparency poses a big challenge in industrial settings, for example when software managers have to provide justification for resource allocation decisions. The lack of transparency therefore directly weakens managerial accountability since such decisions cannot be upheld based on unverifiable automated outputs [10], [11], [12]. Consequently, the persistent gap between model quality and model reliability has become one of the most fundamental obstacles to the practical adoption of SDP. An explanation is useful in practice when it indicates what a developer should inspect, refactor, test, or prioritise, rather than simply identifying a contributing feature. Stability also matters differently by role. For a manager, some variance in an aggregate ranking is a lesser concern. But for a developer, receiving different reasons for the same flagged module across two runs directly erodes trust in the tool.

XAI introduces several techniques that aid human understanding of model prediction [9], [13], [14], [15], thereby addressing this gap. Its goal is to address the black-box problem found in complex ML and DL models, making their decision-making more interpretable for developers as well as stakeholders. SDP was first developed with models whose whole behaviour across an entire dataset could be explained using global explanations. The more recent work, however, has focused on local and instance-level explanations that explain why a specific code unit is predicted to be defective [12], [16], [17]. XAI allows the development team to focus the model on testing and refactoring by identifying which features, such as code complexity, coupling metrics and developer experience, have the most considerable impact on forming a prediction [18], [19]. Within the broader XAI-in-SE literature, SDP is one of the most extensively examined application areas for XAI within the broader field of Software Engineering (SE). Approximately 68% of XAI-related SE research focuses on maintenance activities, including bug prediction [14], [20]. This figure is drawn from Arora et al.'s [20] broader SE-wide XAI survey corpus rather than from an SDP-specific sample. It is reported here as context for the field's overall emphasis rather than as a statistic about the 40 studies in this review's core corpus.

Furthermore, a less-examined yet equally important concern is whether explanations generated by XAI remain stable enough to be trusted. Recent multi-run audits of LIME and SHAP, the two most widely adopted post-hoc explanation tools in SDP, demonstrate that both methods yield inconsistent feature rankings and conflicting contribution directions for identical instances when model configurations or sampling strategies are altered [9], [16], [21]. As a result, if explanations differ between runs, the trust argument that supports XAI adoption fails, regardless of predicted accuracy. Unstable or conflicting post-hoc explanations require careful consideration of how explanation reliability can be improved. One option is to average explanations across repeated runs, while other approaches include explicit stability thresholds, deterministic perturbation sampling, and constrained feature sampling. Feature selection can also be performed before explanations are generated. Another approach is to use interpretable-by-design models, such as the XNAM architecture proposed by He et al., which incorporate interpretability within the model rather than adding it after training. Joint-learning approaches [17] take a similar route by training the predictor and interpreter together. The use of SDP outputs in developer evaluation, workload allocation, and managerial decisions also raises governance concerns. Automated flags should have audit trails, and safeguards are needed to prevent explanations from being used punitively against individual developers. Label noise presents a related concern. SZZ-derived noise is pervasive in the benchmark datasets used throughout this corpus, so XAI explanations based on these labels may create unwarranted confidence. An explanation that appears plausible for a mislabeled instance does not establish that either the model or the explanation is correct. This concern has received far less systematic attention than the well-documented trade-off between accuracy and interpretability, and literature has yet to converge on a consensus solution. Against this backdrop, this paper conducts a review of how XAI methods have been incorporated in SDP, what evidence exists for their effectiveness, and what remains unresolved. We systematically select and analyze 40 relevant studies and present findings, possible research directions, and propose an Actionable and Interpretable Software Defect Framework (AISD-F). The contributions of the study are as follows:

- 1) Presents a PRISMA-guided review of forty peer-reviewed studies on XAI in SDP across five major academic databases. The results are summarised in terms of XAI methods and frameworks, model selection and optimisation, data quality issues, and explanation stability and reliability.

- 2) Explains the inconsistencies between multi-run audits that show instability and single-run evaluations that reflect the dependability of XAI systems.
- 3) Identifies three unresolved issues: cross-project transfer under XAI limitations, temporal concept drift, and human-in-the-loop validation.
- 4) Proposes an AISD-F, a five-stage modular pipeline that integrates socio-technical data, noise reduction, hybrid model selection, dual-layer explanation, and CI/CD.

The rest of this paper is structured as follows. Section 2 provides the related work concerning SDP evolution and XAI techniques along with a summary of directly related studies, and then finally Section 3 describes the methodology. The findings are presented in Section 4, synthesised, contradictions and gaps identified, then the AISD-Framework is proposed in Section 5. Section 6 provides indications for future research, and Section 7 follows with the conclusions of the paper.

2. RELATED WORKS

2.1. SDP Overview

In SQA, research on SDP has evolved through distinct phases. The classical phase is one of the phases that was before the 2010s and relied heavily on statistical learning with simple product metrics such as LOC and McCabe complexity to predict defect density [22]. So, the performance was limited, but at least the models were interpretable by design. Another is the ML era (2010-2017), with classifiers such as Naive Bayes, Support Vector Machines (SVM), and Random Forests (RF), which did improve predictive accuracy [1], [4], [6], but at the cost of transparency. Recent progress in ensemble learning techniques such as RF and XGBoost has been shown to be highly effective in handling imbalanced datasets along with non-linear feature relationships. Concurrently, deep learning models such as CNNs, LSTMs, and Deep Belief Networks have improved prediction accuracy by modelling structural patterns in source code as well as developer histories [1], [6]. Most recently, Large Language Models (LLMs), in particular ChatGPT and Claude, created entirely new opportunities for automated defect detection and repair [22]. As Models have advanced from DT to multi-layered neural networks, internal reasoning has become progressively difficult for humans to interpret and hence the demand for post-hoc explanation methods in research [6].

Throughout the SDP, data quality remained a major constraint throughout all phases. This is because SDP datasets are highly imbalanced, and defective modules represent only a small portion of the samples. This class imbalance can have an impact on both classifier accuracy and the reliability of feature importance estimates [4], [23]. Indeed, this issue is further intensified by class overlaps, where defective and non-defective modules often share similar metric values. According to Gong et al. [24], 70% of datasets that were examined showed overlapping classes, whereas an overlap above 12.5% significantly destabilizes model performance and feature importance ranking. High dimensionality creates another potential issue since several datasets contain redundant or correlated features. Failure to reduce redundant or irrelevant features can lead to model overfitting and XAI explanations misrepresenting the true factors responsible for software defects [2], [14]. Surveys show that an estimated 82-84% of practitioners only value SDP when it can detect and explain defect-associated features, but not when it returns a binary label [14].

2.2. XAI Techniques in SDP

The integration of XAI into SDP plays an important role in making black-box ML models interpretable, enabling developers to trust its capabilities and successfully act upon them. Within the approaches explored, LIME and SHAP are by far the most widely adopted model-agnostic techniques [11], [14], [17], [25]. LIME generates local explanations by using a simpler linear surrogate model around a specific prediction. This is done by using perturbations of input features, such that the decision boundary of the underlying classifier can be estimated [8],[16],[26],[27]. On the contrary, SHAP allocates importance values to features based on their marginal contributions to the final prediction. It provides a global model based on cooperative game theory that is applicable at local and global levels [13], [26], [28], [29]. Empirical studies have demonstrated the practical value of these methods. For example, Jiarpakdee et al. [12] confirmed that practitioners value contrastive, feature-level explanations as important for understanding and interpreting defect model outputs and for decision-making. Based on traditional file-level predictions, the LINE-DP framework used LIME when identifying defective lines within a file while improving both granularity and cost-effectiveness in defect localisation [27]. In the case of DL models, attention mechanisms enable the visualisation of the attention weights so that they can identify which tokens or lines of source code contribute most significantly to a prediction [20], [26]. Nonetheless, in settings where post-hoc explanation is

considered insufficient for transparency, some researchers call for models that are interpretable. These include approaches such as DT and rule induction systems that output human-readable if-then rules without requiring an additional explanatory technique [26].

While XAI becomes widely adopted, the reliability of XAI-generated explanations continues to be challenged. For example, perturbation-based methods such as LIME produce inconsistent explanations for the same instance across runs, which reduces trust in users [9], [12]. When multiple model-agnostic techniques are applied to the same prediction, they tend to disagree on feature rankings and can give conflicting contribution directions for the same feature [21]. Data quality is another reason why the stability of explanation is complicated to attain; class imbalance, label noise, and multicollinearity between software metrics distort feature-importance output in ways that are not always understandable by the user [30], [31], [32], [33]. In order to mitigate instability, several self-explainable architectures, such as eXplainable Neural Additive Models (XNAMs), have been proposed. XNAMs integrate a balance between neural networks and built-in interpretability mechanisms, producing stable feature-impact visualisations without relying on post-hoc methods [34]. However, a long-standing tension persists, where the most accurate models are almost always the most difficult to interpret, and with no standardised approach emerging for balancing this trade-off [25], [26], [35].

2.3. Related Studies

A review of previous surveys and review studies on SDP and XAI is presented in this paragraph, with Table 1 summarizing each study's major contributions. According to Gong et al. [24], class overlap affects data quality and is present in 70% of datasets. They also showed that performance instability is a result of a class overlap ratio greater than 12.5%. In a descriptive analysis employing 106 datasets, Song et al. [23] showed that F-measure and AUC are biased under imbalance and suggested MCC as a more trustworthy substitute. Similarly, Bhandari et al. conducted a meta-analysis on results from 145 studies [2]. They discovered that imbalance and dimensionality lower performance, but they also pointed out that overlap and idea drift are still unexplored.

In terms of methodology, Pachouly et al. [4] traced the development of SDP from manual testing methods to ML and DL methods and suggested a multi-label classification model for both fault detection and severity prediction. Mustaqeem et al. [1] specifically presented a systematic literature review of 79 articles and highlighted the ongoing challenges in dataset labelling and evaluation metrics while recommending adoption of hybrid search-based techniques. In their work, Hasan and Mohi-Aldeen [5] explored DL-based SDP approaches, indicating CNN, LSTM, and Bi-LSTM architectures as effective while also emphasising their constraints due to implementation complexity and data quality. In addition, Jia [22] explored the role of LLMs in the trend based on the F1-score metric and noted that there was no systematic empirical basis for integrating LLMs.

Chamola et al. [36] reviewed the relationship between trustworthy AI and XAI, and introduced a framework based on seven principles, which comply with GDPR requirements. Ortigossa et al. [7] distinguish between intrinsically interpretable models and post-hoc explanation methods such as LIME and SHAP, concluding that the most suitable strategy depends on the deployment environment and stakeholder needs. Mersha et al. [15] then gave a comprehensive taxonomy of XAI by categorising them into two types: Perturbation-based and Gradient-based methods, reviewing their applications across different industries such as healthcare, finance, and autonomous vehicles, and noted that the Transformer-based models have a high potential for interpretability. Arora et al. [20] further mapped interpretability methods, including feature attribution, counterfactuals, and rule extraction, to specific phases of the SDLC. Their review showed that the literature addresses the maintenance phase and identified the lack of standardised evaluation frameworks for assessing explainability in the early stages of software development.

Table 1. Summary of related studies

Ref.	Focus	Contributions/Findings	Limitations
[4] Pachouly et al.	Datasets, validation, AI tools	Highlighted dataset	
		feature gaps; proposed multi-label classification for severity/resource allocation	Tools lack severity prediction; limited actionable support.

Ref.	Focus	Contributions/Findings	Limitations
[1] Mustaqeem et al.	Bibliographic survey (2011–2023)	Suggested validation methods; introduced multi-label classification for broader predictions Showed	Experimental variability; possible study exclusion; limited use of stable metrics Preprocessing
[2] Bhandari et al.	Data quality issues	imbalance/dimensionality degrade performance; recommended MCC	generalisation difficult; narrow within-project focus
[5] Hasan & Mohi-Aldeen	DL-based SDP models (2019–2024)	Identified CNN, LSTM, Bi-LSTM as effective; reviewed metrics/datasets	DL complexity; high dependence on input quality
[22] Jia	Evolution of SDP methods	Compared statistical, DL, and LLM approaches; analysed paradigm shift	LLM integration lacks systematic study; traditional methods resource-intensive
[24] Gong et al.	Class overlap impact	Proposed KNN-based overlap detection; found 70% overlap; improved performance when addressed	No cross-project/JIT contexts; overlap quantification incomplete
[23] Song et al.	Imbalanced learning	Tested 16 methods on 106 datasets; exposed F-measure/AUC bias; recommended MCC	Results vary by procedure; limited generalisability beyond open-source.

Table 1 shows a summary of the existing literature review on SDP, which has examined datasets, validation methods, data quality issues, and DL-based prediction methods [1], [2], [4], [5], [22] - [24]. Although these studies address class imbalance [2], [23] and identify effective DL algorithms [5], they have not systematically investigated the reliability and stability of explanations produced by XAI. This review addresses that limitation by emphasising explanation reliability and the inconsistencies between single-run and multi-run XAI evaluations, and this perspective extends the literature and provides a more comprehensive understanding of explainability in SDP. We further propose AISD-F, a five-

stage modular pipeline to guide future research on explainable SDP. Earlier SE-wide reviews of XAI, including [6], [26], [35], and [36], catalogued explanation techniques across the discipline. However, they did not isolate SDP-specific evidence on explanation stability, nor did they distinguish single-run from multi-run reliability findings. This gap defines the SDP-focused scope adopted here.

3. METHODS

This review followed the PRISMA framework [37] to document the identification, selection, and analysis of literature on Software Defect Prediction (SDP) and Explainable AI (XAI). A systematic review was then conducted to identify, assess, and synthesize studies applying machine learning (ML), deep learning (DL), and XAI techniques to SDP. Relevant publications were retrieved from five academic databases: IEEE Xplore, Scopus, ScienceDirect, Google Scholar, and SpringerLink. These databases were selected for their broad coverage of software engineering and AI research. Registers were used to identify an additional 13 records. Combined with the 83 database records, this gave 96 records in total. After 13 duplicates were removed, 83 records remained for title and abstract screening, as shown in Figure 1. We conducted a Boolean keyword search using AND/OR operators. The exact Boolean search string applied across all databases was: ("software defect prediction" OR "fault prediction" OR "bug prediction") AND ("explainable AI" OR "XAI" OR "LIME" OR "SHAP" OR "interpretability") AND ("machine learning" OR "deep learning" OR "neural network"). Minor syntax adaptations were made for database-specific query parsers. We searched for peer-reviewed journal articles and conference papers from 2020 to 2025.

Study selection followed the four PRISMA stages of identification, screening, eligibility, and inclusion, which are presented in Figure 1. Initially, we retrieved a total of 83 papers in the identification stage. Here, we screened titles and abstracts against the predefined inclusion and exclusion criteria and removed duplicates. Three independent reviewers screened and evaluated studies in parallel. Disagreements were resolved through discussion and consensus, in line with PRISMA recommendations [37]. We did not calculate a formal inter-rater statistic such as Cohen's Kappa. Agreement here came from consensus discussion, not fully independent parallel coding. Because Google Scholar can return non-peer-reviewed and unstable results, each candidate record was cross-checked against IEEE Xplore, Scopus, or SpringerLink before inclusion. Records that could not be

verified as peer-reviewed through this process were excluded at the screening stage. Inclusion criteria were peer-reviewed journal articles or conference papers focusing on SDP or XAI-based SDP as the main topic, published between 2020 and 2025, and in any human language. Titles and abstracts were screened without a language restriction. Full-text eligibility assessment and data extraction, however, required an accessible English-language full text. This is why “not in English” appears as a full-text exclusion reason in Figure 1. Non-peer-reviewed sources such as blogs, technical reports and grey literature; studies not connecting SDP or XAI directly, or lacking enough methodological details to extract data, were excluded. However, two of the papers, [23] and [39], were published in 2019, and we decided to retain them due to the information they have.

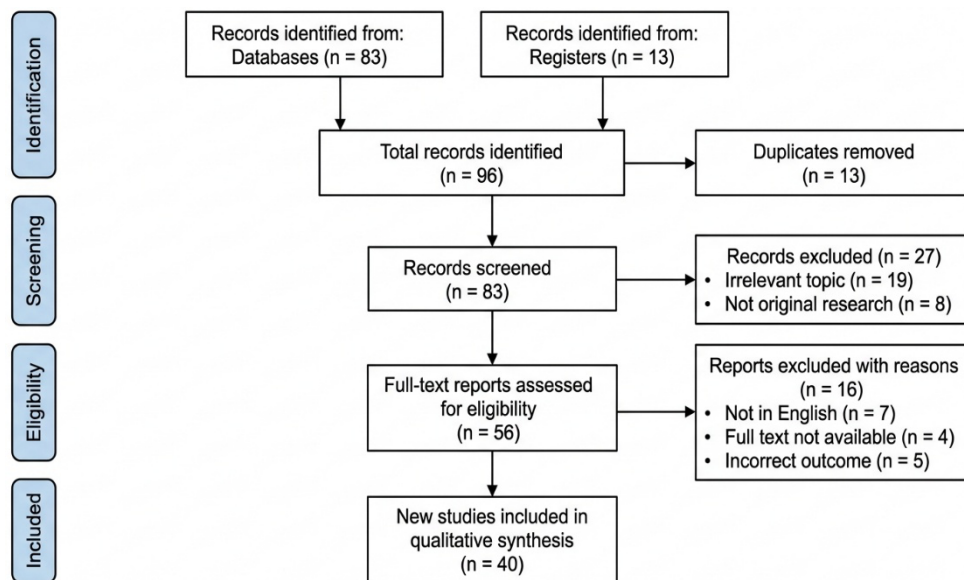


Figure 1. Study selection workflow

After the screening process, 27 records were excluded: 19 as off-topic and 8 as not original research. This left 56 studies assessed for full-text evaluation, consistent with Figure 1. As a result, 40 studies satisfied all quality requirements for inclusion after 16 studies were eliminated at the full-text stage for reasons including non-English language, unavailable full text, and incorrect outcome measures. An organised template was used to analyse each study in order to gather data. Reference and author information were recorded, difficulties were noted, and ML/DL models, datasets, assessment metrics, explainability techniques, conclusions, constraints, and suggested future paths were all

documented. Following extraction, the data were thematically arranged along four dimensions: assessment metrics employed, XAI techniques applied for interpretation, ML/DL methodologies used to handle each difficulty, and data quality challenges. Finally, a qualitative synthesis was used to compare the research methodologies and identify gaps and trends across the reviewed studies. Four thematic dimensions guided the synthesis. These dimensions were derived from the review questions and intended contributions, then refined as recurring patterns emerged during data extraction. For each study, the extraction template recorded nine fields: authors and year, venue, dataset(s), model or technique, XAI method, evaluation metrics, key findings, stated limitations, and suggested future work.

Figure 1 shows the selection and identification process for the study. A systematic screening process was conducted in four stages: identification, screening, eligibility assessment, and inclusion. During identification, 96 records were identified, comprising 83 records retrieved from databases and 13 identified from registers. After removing 13 duplicates, 83 unique records remained for screening. During the screening stage, the 83 records were assessed for relevance to the review topic and originality of the research. Twenty-seven records were excluded, including 19 that addressed topics unrelated to the review and 8 that did not constitute original research or meet the basic inclusion criteria. This left 56 records for full-text assessment. The full texts of the 56 records were assessed against the predefined inclusion criteria. Sixteen reports were excluded: 7 were non-English publications, 4 did not have accessible full texts, and 5 did not report findings relevant to the review. The remaining 40 studies met all inclusion criteria and were included in the qualitative synthesis. Thus, 40 studies were included from the 96 records initially identified. The complete selection process, including identification, duplicate removal, screening, full-text assessment, and final inclusion, is shown in Figure 1.

4. RESULTS AND DISCUSSION

4.1 Results

This section presents the findings of the review, organized into their key thematic categories, while Tables 2 to 6 summarize the reviewed studies.

4.1.1 XAI techniques and explainability frameworks

Among SDLC tasks, SDP has received the most attention in studies of explainability. Most of the reviewed studies applied post-hoc, model-agnostic XAI methods to make defect predictions more understandable to practitioners, with SHAP and LIME being the most frequently used techniques. Such studies are referenced in Table 2, which summarises the studies, and Fig. 3 visualises the distribution of XAI methods. Accordingly, Begum et al. [57] evaluated thirteen ML models on NASA datasets and found XGBoost Regressor to be the most accurate for fault prediction. Dominant features were surfaced using SHAP and LIME, identifying static invocations (nosi) and coupling between objects as the strongest defect indicators. In a similar study, Begum et al. [38] proposed LCNN, a lightweight CNN that was first evaluated on both 1D and 2D architectures using CM1 data for testing. As a result, LIME explanation outputs were much clearer for industry stakeholders than SHAP, particularly for root cause analysis across a range of real-world problems. Gezici and Tarhan [18] similarly applied SHAP force plots alongside ELI5 and LIME to a Gradient Boosting classifier to demonstrate that both local and global explanations could be extracted from an otherwise black-box model. An enhanced set of classifiers was created by combining six XAI techniques, including anchor heatmaps, into their architecture. RF received an F1 score of 0.83, while MLP obtained the best accuracy of 88%. Similarly, Asal and Demir [13] simultaneously used SHAP and LIME in a voting classifier ensemble on the KC2 dataset. Both global feature importance summaries and local explanations for specific forecasts were provided by these two XAI tools.

He et al. [34] introduced XNAMs for Just-in-Time (JIT) defect prediction, which allocate separate sub-networks to every input feature. It reveals three levels of explainability based on gradient-based importance, response curves, and nonlinear interaction effects. When trained on OpenStack and Gerrit as well as other larger open-source projects, XNAMs matched or exceeded leading black-box models while maintaining transparency. Xu et al. [17] took an alternative approach by utilising knowledge distillation to transfer information from a complex CNN predictor to a soft decision tree interpreter through a common fidelity loss function. This joint learning method was able to generate stable local and global interpretations, evaluated across more than 20 datasets. Similarly, Wattanakriengkrai et al. [27] specifically used LIME at the token level within the LINE-DP framework, and the defect prediction was transitioned from file-level to line-level

granularity. Trained on 32 releases to identify defective lines, LINE-DP was not only able to outperform both static analysis and NLP baselines at identifying individual lines but also reduced the cost of inspecting a line. Likewise, Humphreys and Dam [39] employed a self-attention transformer encoder to directly analyse source code semantics. Their model outperformed the prior baselines on Java projects by 4% in F1, and attention matrices were used to show the specific tokens contributing to defect risk.

Research on XAI applications in SE indicates that 68% of the studies focus on the maintenance phase, particularly defect and vulnerability detection. On this note, Li et al. [40] applied SHAP and LIME to graph-based representations of Abstract Syntax Trees. They classified 40 code structures by their contribution to Common Weakness Enumeration (CWE) classification. The resulting feature importance rankings were highly correlated with expert-defined CWE similarities. Ganz et al. [41] also addressed the same problem in their doctoral framework, combining neural code augmentation, causal learning, and dynamic program analysis. They validated their highlighted explanations with actual execution flaws using GNNExplainer and PGExplainer. On a similar note, Siachos et al. [42] used LIME and GNNExplainer to analyse a GAN trained on GitHub and Jira issue reports, identifying keywords that contribute to bug classification predictions. Furthermore, Jude and Uddin [30] demonstrated that stratified splitting outperformed SMOTE for handling class imbalance in defect classification. The stacking ensemble was demonstrated to be the best by achieving the highest accuracy, and SHAP and LIME were used to identify which code metrics impacted model decisions. In parallel, Schulte et al. [29] evaluated LIME and SHAP on more than 3,000 manually rated explanations for bug versus non-bug issue classification. They concluded that SHAP produced less ambiguity and better contextual fit to human annotators, but interpretation remained subjective across annotators.

Table 2. Summary of XAI methods and explainability frameworks

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitation
Gezici and Tarhan [8]	SDP with Gradient Boosting Classifier	Feature effect analysis	NASA (referenced)	Prediction performance	GB Classifier, SHAP	Force plots reveal detailed	SHAP Force Plots	Obscurity of rationale

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitation
					Force Plots	Feature effects		in complex models
Asal and Demir [13]	Enhancing SDP via integrated XAI	Integrated SHAP and LIME in voting	PROMISE/NASA (referenced)	Accuracy	Voting Classifier, SHAP, LIME	Integrating SHAP/LIME improves transparency	SHAP, LIME	Black-box adoption barrier
Xu et al. [17]	Joint learning for prediction and interpretation	Global decision logic	AEEEM, NASA, PROMISE (20 datasets)	MCC, AUC, F1, CD, FI, AI	Joint_CN, Joint_SD, SP-LIME	Joint learning improves accuracy and explanation stability	Joint_SDT	LIME baselines use random sampling
Gezici and Tarhan [18]	Holistic XAI framework for SDP	Sociotechnical gap mapping	KC2 (NASA)	Accuracy, Precision, Recall, F1	RF, GB, NB, MLP, NN, SHAP, LIME	MLP achieved the highest accuracy (88%); RF F1 83%	SHAP, LIME, ELI5	Opaque feature importance
Wattanakriengkrai et al. [27]	Predicting defective lines using XAI	LINE-DP effectiveness	Ant, Camel, Lucene (PROMISE)	Recall@Top20%, IFA	LINE-DP (model-agnostic)	Identifies more defective lines with less effort than baselines	LINE-DP	Qualitative defect analysis still needed
Schulte et al. [29]	Prediction of bug and non-bug issues	Human rater qualitative analysis	Herbold et al. (3,090 issues)	Cronbach's alpha, Spearman's rho	LIME, SHAP (Deep SHAP)	Explanations correlate with human bug definitions	LIME, SHAP	Token leakage in Deep SHAP
Jude and Uddin [30]	Explaining defect classification	Class imbalance and transparency	NASA (CM1, JM1, KC1)	Accuracy, Precision, Recall	SMOTE, GridSearch, Hybrid Algorithms	Improved performance via a hybrid approach	SHAP, LIME	Noise and inconsistencies in datasets

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitation
	using SMOTE							
He et al. [34]	Understanding SDP via explainable NAM	Interpretable classification	OpenStack, Gerrit	Accuracy	Neural Additive Models (NAM)	NAM provides intrinsic interpretability	NAM	High-stakes black-box adoption
Begum et al. [38]	Lightweight CNN for feature identification	1D and 2D CNN comparison	CM1 (PROMISE)	Accuracy, Loss curves	LCNN, LIME, SHAP	LIME outperformed SHAP for root cause analysis	LIME, SHAP	Dataset-specific results
Humphreys and Dam [39]	Explainable deep learning for SDP	File-size classifier baseline	NASA (ant, camel, etc.)	F1, ROC AUC, Precision, Recall	Dummy classifier, file-size classifier	4% F1 improvement over prior baseline (0.67)	Attention-based interpretability	Data leakage in standard datasets
Li et al. [40]	Vulnerability factors via model-agnostic XAI	CWE type distribution	Juliet Test Suite, OWASP, Draper	Accuracy, Rank correlation	SHAP, LIME, RFs	High-impact factors for CWE types identified	SHAP, LIME	CWE distribution varies by dataset
Ganz et al. [41]	Vulnerability localization via explainable DL	Robustness and context sensitivity	Devign, ReVeal	Contrastivity, Sparsity, Efficiency	GNNExplainer, PGExplainer, Graph-LRP	Detailed accuracy vs. sparsity comparison reported	GNNE explainer, PGExplainer	Wrongly labelled samples in Devign
Siachos et al. [42]	Bug prediction using Graph Neural	Graph-based textual representation	GitHub (450k), Jira (1.4m) issues	Train/test document stats	GAT, Word Embeddings	GAT with 10-unit hidden layer achieves strong classification	Multi-modal XAI (referenced)	Issue titles disregarded

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitation
	Networks							
Begum et al. [57]	Identification of software defects using ML/XAI	XGBoost and SHAP application	PC1, Baseline, CM1, JM1 (PROMISE)	RMSE, R2, EV, D2 score	XGBR, SMOTTEch, Python	XGBR regressor demonstrates superior performance	SHAP	RMSE values remain high for some models

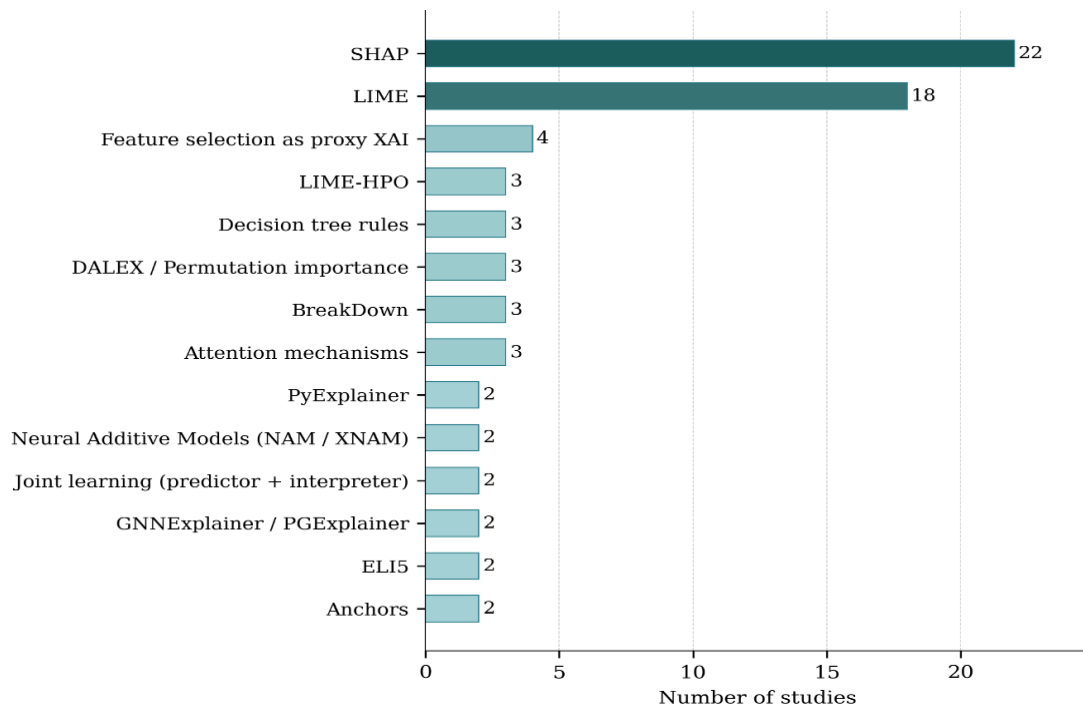


Figure 3. XAI methods frequency across studies

4.1.2 Explanation Reliability and Stability

Through the reviewed studies, stability of explanations has emerged as a main concern with regard to whether post-hoc XAI methods produce explanations stable enough for practical use. Here, tools such as LIME can provide different explanations for the same instance across multiple runs due to sampling variance. To achieve this, Awal and Roy [9] proposed rule-based XAI tools, including PyExplainer and LIME, for just-in-time defect prediction, clone detection, and code review tasks. By employing their EvaluateXAI framework, they showed how often these tools produced contradictory or anomalous

explanations when applied to the same instance across multiple runs. This demonstrates that the unreliability of XAI outputs is not an isolated edge case but a measurable and systematic problem. It suggests that resolution may lie in formal methods rather than ad hoc fixes. Equally, Shin et al. [16] examined the explanation outputs for LIME-HPO and BreakDown on historical Java project data and found that the outputs varied substantially when sampling methods, classifiers, or cross-version configurations changed. Different settings produced different rankings of features for the same code instances. This instability was confirmed by a user study and directly hindered developers' understanding.

Extended reliability analysis was conducted by Lee and Lee [21] on 300,000 local explanations generated by LIME, SHAP, BreakDown, and Anchors. The study showed that feature importance rankings and impact directions were highly inconsistent, with LIME exhibiting the greatest divergence from other prediction methods. Their findings cautioned practitioners against using these tools interchangeably without manual validation of outputs. Similarly, Jiarpakdee et al. [12] examined the issue in terms of practitioner trust. By comparing with state-of-the-art interpretations, they demonstrated that LIME and BreakDown can provide accurate yet diverse local explanations for flagged modules and proposed LIME-HPO as a refined variant with improved local model fit. They classified instance-level explanation as a key requirement not only for usability but also for compliance with transparency requirements under regulations such as GDPR. Table 3 summarises the studies on the reliability and stability of explanation.

Table 3. Summary of the reliability and stability of the explanation

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitation
Awal and Roy [9]	Evaluate reliability of rule-based XAI	Rule-based XAI stability	JIT Defect, Clone Detect ion, Code Review	Accuracy, F1, AUC, Stability	PyExplainer, LIME, SMOTE, AutoSpearman	Rule-based XAI frequently fails to produce reliable	PyExplainer, LIME	Inconsistency in explanation output

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitation
						explanations		
Shin et al. [16]	Empirical study on stability of model-agnostic XAI	Explanation across settings	Java projects (historical data)	Hit rate, Rank difference	LIME-HPO, BreakDown, imbalanced-learn	LIME-HPO and BreakDown inconsistent under different settings	LIME-HPO, BreakDown	Limited number of techniques evaluated
Lee and Lee [21]	Model-agnostic technique comparison	Local explanation representativeness	Yatish et al. (32 releases)	Spearman's rho, Kendall's rank	RF, GBM, MLP, XGBT, Anchors	LIME diverges most from other methods	Anchors, LIME	Dataset cannot represent all defect types
Jiarpakdee et al. [12]	Reliability and practitioners' perception of XAI	Trustworthiness and reliability	Yatish et al. (affected releases)	Scott-Knott ESD test	LIME, BreakDown, LIME-HPO	LIME-HPO improves explanations consistency	LIME, BreakDown	Small Mechanical Turk sample (n=20)

4.1.3 Data Quality Challenges

XAI performance in SDP tracks data quality closely: class overlap and ranking instability constrain both prediction accuracy and explanation reliability, and many of the reviewed studies target these directly. Table 4 summarises their results, and Figure 4 reports how

often each dataset appears: NASA datasets account for roughly 14 of the 40 papers. Bommi and Negi [43] ran a multi-stage preprocessing pipeline on the JM1 dataset: scaling, dimensionality reduction, rebalancing, after which oversampling traded raw accuracy for a higher detection rate of actual defects; LIME and counterfactual explanations then conveyed the resulting model's decisions. Mehmood et al. [25] tested feature selection across ten classifiers on five NASA benchmark datasets, improving accuracy by roughly 5% over standard methods, confirmed with a paired t-test, with K-NN, BP-NN, and SVM performing best under the reduced feature set. Olaleye et al. [31] tackled the same multicollinearity problem in the KC1 dataset with Variance Inflation Factor analysis followed by PCA; their MLP-based approach reached near-perfect classification, and LIME traced the transformed features to thresholds associated with high defect risk. Mustaqeem et al. [44] extended this preprocessing logic with SPAM-XAI, combining SMOTE, PCA, and MLP with LIME-based explanation, reaching 98.65% accuracy on NASA datasets while cutting computational overhead. Preprocessing decisions frequently compromise raw accuracy in favour of improved defect identification or reduced overhead throughout this cluster.

Wu et al. [32] investigated label-level data quality. Their manual reassessment of five widely used security bug report benchmarks identified hundreds of misclassified vulnerabilities. After correcting the labels, data quality had a greater effect on performance than model architecture, with simple text classifiers outperforming more complex frameworks trained on the original noisy datasets. Meyer et al. [45] examined data quality from a fairness perspective and proposed dataset multiplicity to assess prediction stability under changes in training data. Using linear models with label errors, they found that small data perturbations affected predictions unevenly, with some demographic groups experiencing greater instability than others. Radliński [46] evaluated 16 predictive models across 40 data variants of the ISBSG dataset. Larger samples performed better than filtered subsets, challenging the assumption that removing low-quality records necessarily improves predictive performance. Malhotra and Khan [33] targeted the imbalance side directly with OpTunedSMOTE, built on Optuna with Tree-structured Parzen Estimators to automate SMOTE hyperparameter tuning across seventeen imbalanced datasets and five classifiers; Friedman's test confirmed improved ROC-AUC and MCC scores over fixed-configuration SMOTE, at the cost of higher computational demand. Taken together, these four studies suggest the field's model

comparisons may be measuring label noise and sampling artefacts as much as algorithmic merit.

Table 4. Summary of data quality challenges

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitations
Bommi and Negi [43]	Establishing a strong SDP baseline	Preprocessing vs. models	PROMISE	Accuracy, Recall, Precision, F1	PCA, LR, NB, KNN, SVM, RF	Pipeline helps standardise future comparison	LIME (referenced)	Non-reproducible academic studies
Mehmood et al. [25]	Improve SDP accuracy using ML	Feature selection and data quality	PROMISE (JM1, CM1, KC1, PC1, KC2)	Accuracy, Specificity, Sensitivity	Minitab (T-test), WEKA, SVM, C4.5	K-NN, BP-NN, and SVM provide superior results	Feature selection framework	Redundant and irrelevant features
Olaleye et al. [31]	Multicollinearity mitigation in KC1	VIF analysis and defect localization	KC1, KC1_D (NASA)	VIF, MLP performance	MLP, LIME, PCA, VIF Analysis	VIF analysis identifies severe multicollinearity	LIME	Design specific to KC1
Mustaqeem et al. [44]	Trustworthy hybrid model (SPAM-XAI)	Overfitting and transparency	NASA (CM1, PC1, PC2)	Precision, Recall, F-measure, AUC	SMOTE, PCA, MLP, LIME	Accuracy up to 97% on CM1 with PCA reduction	LIME	Limited independence in base models
Meyer et al. [45]	Impact of unreliable data on predictions	Underspecification and fairness	National Loan-Level Dataset (2019)	Fairness-accuracy trade-off	Hyperparameter optimization	Dataset multiplicity produces varied predictions	Datasets for datasets	Underspecification in ML models
Wu et al. [32]	Impact of label correctness	Label noise	Chromium, Ambari, Camel	Recall, Precision, F1, G-measure	RF, SMOTE, Hyperparameter	Performance substantially higher on	Manual annotation	High manual annotation cost

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Quantitative Results	XAI	Limitations
	ess for SBR		Derby, Wicket		parameter tuning	corrected datasets	analysis	
Radliński [46]	Data quality impact on effort prediction	ISBSG filtering strategies	ISBSG 2020 R1 (1,242 projects)	AE, MAE	enet, nnet, xgbTree, Wilcoxon test	enet/nnet performs best on full dataset	Data quality rating	Arbitrary decisions in attribute filtering
Malhotra and Khan [33]	Automated hyperparameter tuning of SMOTE	Optuna Framework	17 imbalanced datasets	ROC-AUC, MCC, Runtime	OPTUNA, TPE algorithm, MLP, RF	Optimising SMOTE improves ROC-AUC	Ranking classification	High misclassification costs

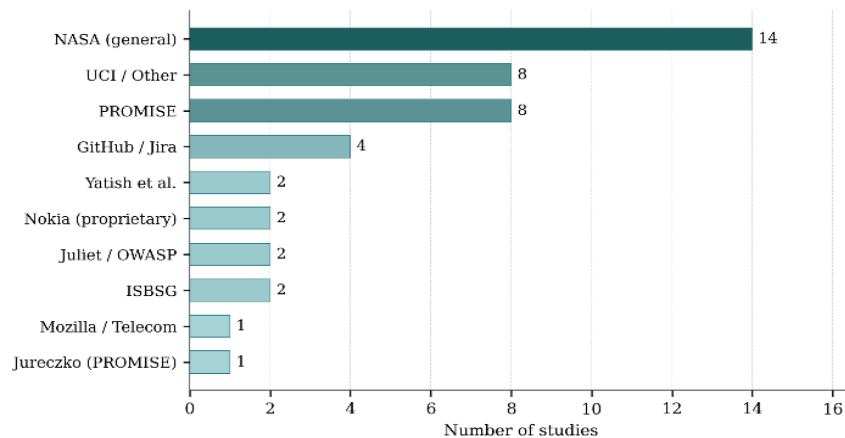


Figure 4. Datasets usage frequency across studies

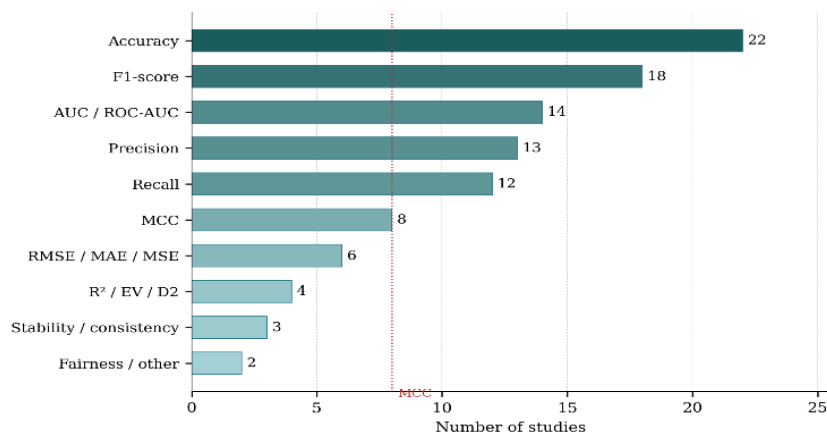


Figure 5. Evaluation of metrics frequency across studies

Figure 5 reports how often each evaluation metric appears across the reviewed studies' datasets, which covers Accuracy, F1, AUC, MCC, Precision, Recall, and RMSE. Here, Accuracy leads the set at 22 studies, F1-score follows at 18, and MCC trails at only 8. Many studies flag MCC as the more reliable choice for imbalanced datasets. Yet the count in Fig. 5 reveals a different picture, remaining one of the least-used metrics in the pool. The red dotted line marks MCC's position within the distribution.

4.1.4 Hyperparameter tuning and model optimisation

In an XAI-based SDP, hyperparameter optimisation (HPO) plays three central roles in improving model performance, ensuring consistent explanations, and preventing fitting issues such as underfitting or overfitting [14], [6]. Across the studies reviewed, some focused only on HPO with the objective of improving defect prediction accuracy. Table 5 summarises these studies. Al-Hadidi and Hasoon [47] used grid search with cross-validation for fine-tuning the XGBoost algorithm across NASA MDP datasets, showing that optimised ensemble learning achieved accurate classifications. They point out that reliability in safety-critical domains is required; therefore, fine-tuning is also required for maximum accuracy. Similarly, Al-Fraihat et al. [48] proved that tuning ensemble classifiers with cross-validation improves accuracy and reduces costs associated with defect discovery in a late-phase software development cycle; they evaluated this on the JM1 dataset.

Furthermore, Das et al. [49] presented a method of four hybrid bio-optimized algorithms which combined the Sparrow Search Algorithm (SSA) with an enhanced Grasshopper Optimisation Algorithm for HPO of ANN and XGBoost models using thirteen NASA datasets. Their evaluation revealed an average accuracy of 94% that outperformed baseline metaheuristic algorithms. Raji et al. [50] also proposed SDSGA, a deterministic genetic algorithm using elitist-like selection and elevated mutation probability; the model achieved a comparable accuracy on CNN and RF models when using fewer function evaluations than the Bayesian Optimisation framework. Antal-Vaida [51] evaluated GridSearch and RandomizedSearch on a UCI financial dataset and found modest improvements for DT and KNN while cautioning that tuning benefits must be weighed against computational cost.

Table 5. Hyperparameter tuning and model optimization

Ref.	Objectives	Focus	Datasets	Metrics	Techniques / Tools	Results	XAI	Limitations
Al-Hadi and Hasoon [47]	SDP using XGBoost with hyperparameter tuning	Data cleaning and accuracy	NASA MDP (CM1, MC1, PC1)	TP, FN, FP, TN	XGBoost, data cleaning	High categorisation rates achieved	Not explicitly stated	Erroneous/duplicate entries
Al-Fraihat et al. [48]	HPO for bug prediction	CVParameter selection	JM1 (NASA)	Accuracy	Ensemble classifier, CVParameter	Tuning improves predictive accuracy	Not explicitly stated	24% duplicated instances in JM1
Das et al. [49]	HPT in ML models for SDP	Hybrid bio-optimised algorithms	13 NASA defect datasets	Accuracy, Runtime	BoA, XGB, ANN	Average accuracy 94% with bio-optimisation	Improvisation explanation	Scalability constraints
Antal-Vaida [51]	Basic tuning methods for classification	General tuning study	UCI (Credit card clients)	Accuracy	Scikit-learn tuning	Modest gains for Decision Trees and KNN	Not in excerpt	Specific to credit dataset
Raji et al. [50]	SDSGA for hyperparameter tuning	Metaheuristic efficiency	5 UCI datasets (Breast Cancer, etc.)	Accuracy	BO, SDSGA, GA, PSO, BBO	98.246% accuracy on Breast Cancer	Parameter combinations	Dependent on initial population

4.1.5 Model selection and comparative studies

The main objective of XAI is to address the black-box nature of complex ML/DL models by improving the transparency and interpretability of their decision-making. However, several studies evaluated these methods without focusing on XAI, but they still contributed evidence on which model families perform best under specific dataset conditions. Table 6 summarises these studies, while Figure 6 shows the frequency of models used across them.

For cross-company defect density prediction, Haldar and Capretz [52] employed regression-based tree algorithms on the ISBSG dataset. They didn't use the SHAP tool for their main research objective, but rather to explain feature contributions for project managers. In addition to this, ExtraTreeRegressor and CatBoost showed the best results. Similarly, Madeyski and Stradowski [53] designed LA2SDP from test data of the Nokia 5G system to detect patterns in SDP with high accuracy. Their findings show that CatBoost and RF achieved Matthews Correlation Coefficient (MCC) values above 0.3 without relying on code metrics. They offer a lightweight solution to standard SDP pipelines often offered in industrial settings. Polu [11] also trained Gradient Boosting and DNNs with a dataset combining code complexity, team dynamics, and financial indicators. Results showed that those new project failures could be predicted as early as eight weeks prior to the actual event occurring. In a similar note, Tadapaneni et al. [54] compared Naïve Bayes with LSTM and DNN on 10,885 NASA instances and found that DNN produced the lowest error rates. Albattah and Alzahrani [55] used an integrated Java bug dataset containing more than 47,000 classes to evaluate eight ML and DL models. Among the evaluated models, LSTM achieved the best performance by modelling code-quality patterns for early defect detection.

Chai et al. [28] examined Social Network Analysis (SNA) metrics, including betweenness and pWeakC(out), as alternatives to conventional size measures such as LOC. On PROMISE datasets, their SHAP analysis showed that these dependency-based measures provided better prediction performance while also offering greater structural interpretability. Other studies similarly examined whether simpler feature sets could outperform more extensive feature engineering. Esteves et al. [19], for example, found that US-XGBoost models trained on one to three SHAP-selected features from individual Jureczko datasets

often outperformed established baselines. Their findings suggest that principled selection of a small feature set can rival, and in some cases surpass, the performance of more elaborate feature engineering approaches. Chmielowski et al. [56] equally found that decision tree classifiers trained on Mozilla and internal telecommunications data generated human-readable rules with the same accuracy as opaque models for bug severity classification. This leads to the conclusion supporting inherently interpretable models in settings where transparency is strictly required. A study by Stradowski and Madeyski [10] used DALEX-based permutation importance to determine what stakeholders can change with respect to XAI deployment, finding the test date and responsible organisation as the most actionable features in Nokia's 5G testing process. Their framework provided a practical model for handling XAI adoption in an industry scenario, as they categorised stakeholders by information needs.

Table 6. Summary of model selection approaches

Ref.	Objective	Focus	Datasets	Metric	Techniques / Tools	Quantitative Results	XAI	Limitation
Haldar and Capretz [52]	SDP from cross-company project-level data	Regression-based tree algorithms	ISBSG D&E 2021 R2 (10,600 records)	MAE, MSE, RMSE, R2	ExtraTree, CatBoost, RF, XGB, SHAP	ExtraTree/CatBoost best performing	SHAP	High rate of missing values in ISBSG
Madeyski and Stradowski [53]	Lightweight alternative to standard SDP	System-level test repository	Nokia Test Repository (QC1–QC6)	MCC, Accuracy, Precision, AUC	CatBoost, LGBM, RF, NB, DALEX	CatBoost achieved MCC > 0.3	DALEX	Proprietary data limits generalisation
Polu [11]	Predict software project failure risks	Feature selection and model comparison	Labeled failure datasets	Accuracy, Error metrics	LASSO, PCA, RF, GBM, DNN	Failures flagged up to 8 weeks in advance	PCA for transparency	Potential for overfitting
Chai et al. [28]	Alternatives to size metrics	SNA	PROMISE	AUC-ROC	Understand (v5.1), Ucinet	SNA metrics are effective alternatives	SHAP	Limited generalizability to

	for explainable SDP		(jEdit, etc.)	ACC, MCC	AutoSpearman	to size metrics		commercial projects
Chmielowski et al. [56]	XAI in bug classification	Security vs. non-security issues	Mozilla Defect Database, internal data	Accuracy	KNN, LR, NB, SVC, XGBoost, Decision Tree	Tree-based models provide rules without accuracy loss	Decision Tree rules	Trade secrets limit internal data disclosure
Esteves et al. [19]	Understanding ML defect predictions	Feature importance and Scott-Knott ESD	Jureczko (TOMCAT, ANT, LOG4J)	F1, AUC	XGBoost, US-XGBoost, XGBoost, Scott-Knott ESD	US-XGBoost often outperforms complex models	Permutation importance	Aggregated metrics hinder performance
Stradowski and Madeyski [10]	Applied ML interpretability in vivo	Stakeholder trust in Nokia 5G	Nokia 5G system testing	MCC	DALEX, Permutation importance	Test date and organisation identified as most impactful features	DALEX	Industrial data quality varies
Tadapaneni et al. [54]	SDP based on ML/DL	Feature extraction	NASA (10,885 instances)	22 Features	Data exploratory analysis	DNN achieves lowest error rates	Not explicitly detailed	Not stated
Albattah and Alzaharani [55]	SDP empirical approach	Bug prediction on dataset assessment	Java bug database (Ferenc et al.)	Accuracy	ML and DL techniques	LSTM most effective for early defect detection	Not in excerpt	Dataset available on request

As shown in Figure 6, ML/DL tagged on the right-side show that traditional learners, such as RF, XGBoost, and SVM, dominate the top of the frequency ranking, while DL models, such as MLP, LSTMs, and CNNs, cluster in the middle tier. This supports the hybrid engine argument in our proposed AISD-F explainable SDP model. To make the task-level composition of the corpus explicit, Table 7 reclassifies all forty included studies by task type, prediction granularity, XAI technique, and validation type. Ten of these studies

address tasks adjacent to core SDP, such as vulnerability localisation, issue classification, project-failure prediction, or testing-effort prediction, and are retained only for their XAI-methodology relevance. Only four studies report validation beyond automated metrics: two through industrial deployment and two through human raters. This concentration reinforces the human-in-the-loop and industrial-validation gaps discussed in Section 4.2.

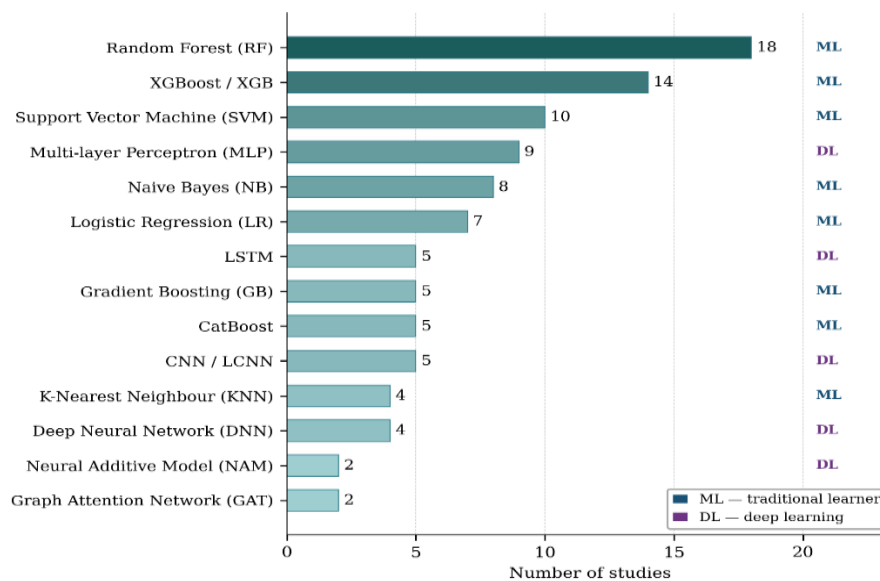


Figure 6. ML/DL model frequency across studies

Table 7. Task, granularity, and validation taxonomy

Ref.	Task type	Granularity	XAI technique(s)	Validation type
[8]	Core SDP	File-level	SHAP force plots	None reported
[9]	Core SDP	Commit-level / JIT	PyExplainer, LIME	None reported
[10]	Core SDP	System/test-level	DALEX permutation importance	Industrial (Nokia 5G)
[11]	Adjacent: project-failure prediction	Project-level	PCA (for transparency)	None reported
[12]	Core SDP	File-level	LIME, BreakDown, LIME-HPO	Human raters (Mechanical Turk, n=20)

Ref.	Task type	Granularity	XAI technique(s)	Validation type
[13]	Core SDP	File-level	SHAP, LIME	None reported
[16]	Core SDP	File-level	LIME-HPO, BreakDown	Human user study
[17]	Core SDP	File-level	Joint_SDT, SP-LIME	None reported
[18]	Core SDP	File-level	SHAP, LIME, ELI5 Permutation	None reported
[19]	Core SDP	File-level	importance (US-XGBoost)	None reported
[21]	Core SDP	File-level	LIME, SHAP, BreakDown, Anchors	None reported
[25]	Core SDP	File-level	Feature-selection Framework	None reported
[27]	Core SDP	Line-level	LINE-DP (model-agnostic)	None reported
[28]	Core SDP	File-level	SHAP (SNA metrics)	None reported
[29]	Adjacent: issue classification (bug vs. non-bug)	Issue-level	LIME, SHAP (Deep SHAP)	Human raters (n ≈ 3,090 issues)
[30]	Core SDP	File-level	SHAP, LIME	None reported
[31]	Core SDP	File-level	LIME, PCA, VIF analysis	None reported
[32]	Adjacent: security bug report label quality	Issue-level	Manual annotation analysis	None reported
[33]	Core SDP	File-level	Ranking classification (SMOTE tuning)	None reported
[34]	Core SDP	Commit-level / JIT	NAM (self-explainable)	None reported
[38]	Core SDP	File-level	LIME, SHAP	None reported
[39]	Core SDP (pre-2020, background)	File-level	Attention-based	None reported

Ref.	Task type	Granularity	XAI technique(s)	Validation type
[40]	Adjacent: vulnerability (CWE classification)	Code-structure level	SHAP, LIME	None reported
[41]	Adjacent: vulnerability localization	Line-level	GNNExplainer, PGExplainer	None reported
[42]	Adjacent: issue classification	Issue-level	LIME, GNNExplainer	None reported
[43]	Core SDP	File-level	LIME (referenced)	None reported
[44]	Core SDP	File-level	SMOTE, PCA, MLP, LIME	None reported
[45]	Adjacent: dataset reliability/fairness (non-SDP dataset)	Not applicable	Datasheets for datasets	None reported
[46]	Adjacent: testing- effort prediction	Project-level	Data-quality rating (not XAI)	None reported
[47]	Core SDP	File-level	Not explicitly stated	None reported
[48]	Core SDP	File-level	Not explicitly stated	None reported
[49]	Core SDP	File-level	Improvisation explanation	None reported
[50]	Adjacent: general metaheuristic tuning (non-SDP dataset)	Not applicable	Parameter combinations	None reported
[51]	Adjacent: general classification tuning (non-SDP dataset)	Not applicable	Not in excerpt	None reported
[52]	Core SDP	Project-level	SHAP	None reported
[53]	Core SDP (lightweight alt.)	System/test- level	DALEX	Industrial (Nokia 5G)
[54]	Core SDP	File-level	Not explicitly detailed	None reported
[55]	Core SDP	File-level	Not in excerpt	None reported

Ref.	Task type	Granularity	XAI technique(s)	Validation type
	Adjacent: security vs. non-security issue classification			
[56]		Issue-level	Decision Tree rules	None reported
[57]	Core SDP	File-level	SHAP	None reported

Table 8 further synthesizes how each RQ and contribution stated in the Introduction is answered, and where the corresponding evidence and discussion appear.

Table 8. RQ-by-RQ synthesis

RQ / contribution	Finding	Where addressed
RQ1: What XAI methods and frameworks, data-quality issues, and model-selection strategies appear in SDP, and how stable/reliable are their explanations?	LIME and SHAP dominate; multi-run audits show both produce inconsistent rankings under changing configurations, contradicting single-run reliability claims.	Section 4.1A-D
RQ2: Why do single-run evaluations report reliable explanations while multi-run audits report instability?	The difference is mainly methodological. Single-run studies evaluate goodness-of-fit for one configuration, whereas multi-run audits examine variation across hundreds of repeated runs.	Section 4.1B and 4.2
RQ3: What research gaps remain unresolved in XAI-for-SDP?	The review identifies three main gaps: limited cross-project transfer under XAI constraints, temporal concept drift, and the lack of human-in-the-loop validation.	Section 4.2 and 4.4
RQ4: How can these findings be translated into a practical framework?	AISD-F is proposed as a five-stage conceptual pipeline covering data quality, hybrid model selection, dual-layer explanation with a human-in-the-loop	

checkpoint, and CI/CD integration. The framework remains conceptual and has not yet been empirically validated.

4.2. Discussion

This review has systematically reviewed 40 studies on SDP and its XAI techniques' effectiveness. During the review process, three themes emerged with consistency across the studies. First, interpretability has moved from being a desirable feature to a practical requirement. However, industrial adoption is still restricted since practitioners cannot act on a binary prognosis without comprehending the reasons behind that prediction [8], [12], [13], [18], [34]. In controlled settings, ML and DL model accuracy frequently surpasses 80-90%. Feature-level explanations that indicate particular remedial actions, which module characteristics to refactor, and which developer process to signal are what are needed rather than higher AUC scores. Second, data quality has more restrictions on explanation validity and prediction reliability than model architecture. For instance, class imbalance, multicollinearity, label noise from the SZZ algorithm, and class overlap affect most of the public SDP datasets [9], [32], [33], [53]. Preprocessing decisions such as choice of imbalance handler, correlation filter, and feature selection strategy have been shown to have as much effect on final performance as the choice of learner [21], [30], [33]. Third, the granularity of prediction is shifting drastically. In this case, file-level classification is increasingly viewed as too coarse for practical defect management. This approach drove the field to JIT prediction at the commit level, followed more recently by line-level localisation that reduces the inspection effort required from developers [27], [34].

Likewise, the studies reviewed reveal several points of real disagreement that should be acknowledged. The most crucial concern is the reliability of XAI. Few studies propose LIME and SHAP as consistent and trustworthy tools for building practitioner confidence [8], [19], [30]. However, even with rigorous multi-run audits, studies show that both methods can produce inconsistent feature rankings and conflicting contribution directions when random draws of sampling methods, classifiers, or when running different cross-version configurations [9], [16], [21]. This difference has been recognised and is attributed primarily to methodological variations. This is because studies reporting consistency often evaluate only a single run against goodness-of-fit criteria, while those reporting instability require variance across hundreds of runs [9], [21]. If explanations

differ after each time a model is retrained, the case for trusting XAI results fails regardless of predictive accuracy.

Another tension also relates to the validity of data for benchmarked datasets, especially since several studies still consider NASA MDP and PROMISE benchmarks as gold standard [25], [48], [57]. Other studies have identified regular issues in these datasets, such as labels being nearly 50% noisy, duplicate instances, and feature sets which no longer represent modern codebases [31], [32]. Because of this, the value of size measurements like LOC is still debatable. LOC has historically been regarded as one of the primary complexity markers. SNA dependency measurements, however, offer better prediction performance and more comprehensible structural signals, according to current data [28]. Hyperparameter tweaking is the third source of tension. Effort-prediction research casts doubt on the idea that it is necessary for dependable performance [33], [48], since bigger, unbalanced samples may perform better than smaller, cleansed datasets [46], [33]. Benefits from tuning seem to be algorithm-specific rather than general.

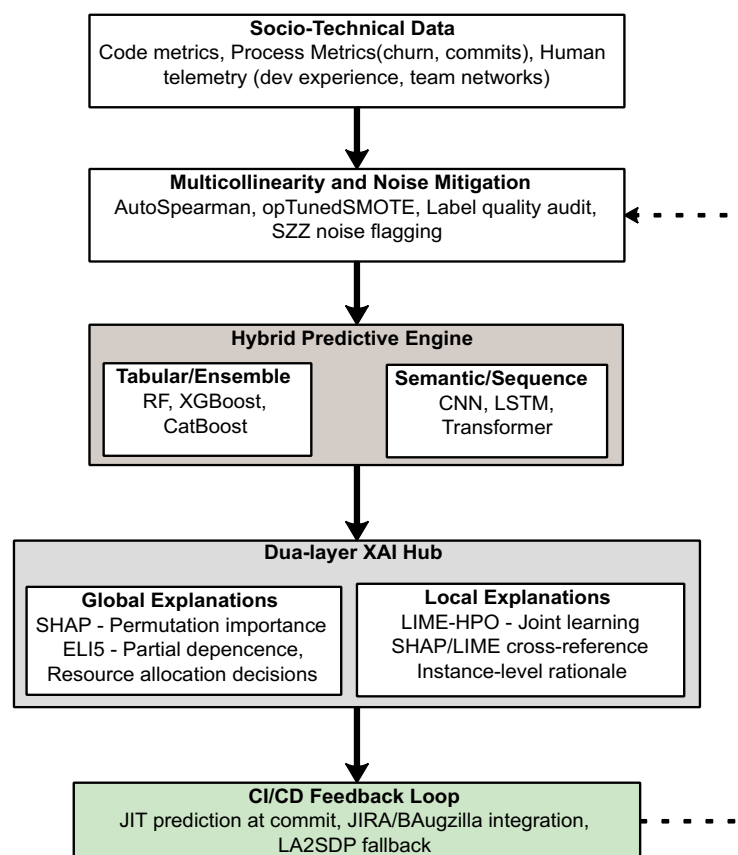


Figure 7. AISD-F pipeline

Additionally, our results demonstrate that three research gaps are still unmet in all of the analysed papers. First, human-in-the-loop validation is not included. Although there is no controlled empirical proof that XAI-assisted defect prediction results in faster or more accurate bug repairs than unassisted practice, studies assume that better explanations increase developer productivity [18], [45], [29]. There has been no scientific testing of the link between explanation, trust, and better results. Second, the majority of research ignores how code patterns, defect definitions, and team configurations change throughout the course of a project, treating project data as static [30], [34]. A model trained on one phase of historical data may be poorly calibrated to a later phase without explicit drift detection and updating, a possibility these studies don't account for. Third, cross-project generalisation stays fragile. Models that perform well within-project often degrade when applied to a different repository, and what causes that degradation, or what conditions allow successful transfer, is still poorly understood [28], [52], [53].

4.3. Proposed AISD-F framework

This section presents the proposed AISD-F framework, shown in Fig. 7. Our framework is derived from the key findings identified from reviewed studies. Collectively, these studies support a modular pipeline in which data quality and explainability are treated as first-class concerns at every stage rather than as post-processing additions.

The AISD-F framework consists of five components, organised as a five-stage modular architecture that supports the end-to-end SDP workflow, from data ingestion to CI/CD integration. The first module is a socio-technical data layer that integrates process metrics, code churn, commit frequency, human telemetry, developer experience, and team communication networks, alongside static code metrics [28], [30]. This layer reflects evidence showing that human and organisational factors are often more influential predictors of failure than source code attributes alone [11], [53]. The second stage is a multicollinearity and noise-mitigation hub applied before model training. AutoSpearman is used to remove redundant correlated features, and label quality is explicitly assessed rather than assumed [9], [28], [31]. The framework identifies this as a known reliability constraint and suggests sensitivity analysis across the labelled subset where SZZ-derived labels are the only accessible choice [32], [53]. The third element is a hybrid predictive engine that chooses between DL architectures like CNN and LSTM for sequence or semantic representations of source code and ensemble ML, RF, XGBoost, and CatBoost

for tabular project metrics [11], [30], [44], and [54]. The size of the dataset influences the model selection process. For instance, DL architectures are unsuitable for the small within-project datasets typical of many SDP benchmarks, where simpler models consistently match or exceed their performance [18], [46].

The fourth component is the dual-layer XAI hub, which addresses the stability problem directly. Here, global explanations via SHAP or permutation importance are generated for project-level resource allocation decisions. Local, instance-level explanations are generated for individual module predictions using LIME-HPO or, where stability is critical, a joint learning framework that trains the predictor and interpreter simultaneously rather than applying post-hoc tools to a separately trained model [12], [17].

Explanations from LIME and SHAP are cross-referenced here. Where SHAP and LIME disagree on feature direction, SHAP is provisionally prioritised for numeric outputs, based on evidence of lower ambiguity in human-annotated comparisons [29]. This default remains configurable and requires further validation before industrial adoption, and the disagreement is presented to the practitioner rather than suppressed. This is based on a single comparative study, not a broader consensus, so it should be read as a provisional default rather than a general finding. Within this dual-layer hub, a human-in-the-loop review checkpoint is included explicitly. Flagged disagreements between global and local explanations, or between LIME and SHAP outputs, are routed to a practitioner for review before action, rather than resolved automatically by a fixed rule. This directly addresses the absence of human-centered validation identified as a major gap throughout this review, while keeping AISD-F at five stages by embedding the checkpoint within the existing dual-layer hub rather than adding a sixth. In the same vein, the fifth component is a CI/CD feedback loop that integrates JIT predictions at the point of commit, mapping flagged changes to issue tracker entries: JIRA, Bugzilla, to enable immediate intervention [11], [34], [53]. Where commit history is too noisy or unavailable, the LA2SDP lightweight alternative predicts test failures from test repository data rather than mining defect-inducing commits [10], [53].

4.4. Possible research directions

This subsection presents six research directions that emerged from the gaps and tensions identified in this review. In terms of human-in-the-loop evaluation, the most

pressing unaddressed question in the field is whether XAI-generated explanations improve developer outcomes in practice. In this case, controlled experiments like randomised or quasi-experimental studies, comparing defect resolution time and accuracy with and without XAI assistance, are needed before the practitioner's value of explanation tools can be claimed rather than assumed [18], [29], [45]. Such studies would also clarify which explanation types- local feature attributions, counterfactuals, and rule-based summaries are most useful for different developer roles and tasks. Another aspect is the explanation of stability standards. The instability of LIME, SHAP, and related tools under different configurations is well documented but unresolved [9], [16], [21]. Future work should establish minimum stability thresholds, defined in terms of rank consistency and directional agreement across runs, as evaluation criteria alongside predictive metrics. Joint learning frameworks are the most promising current response to this problem [17], but they have been validated on a narrow set of datasets and require broader empirical assessment.

Temporal modelling and concept drift equally need to be addressed. SDP models are almost universally evaluated on static snapshots of project history. Incorporating explicit drift detection mechanisms like sliding window retraining, change point detection, and version-aware feature engineering would bring experimental designs closer to the conditions under which models would operate in a CI/CD environment [30], [34]. Longitudinal studies tracking model calibration across multiple release cycles are particularly lacking. In the same vein, cross-project and cross-organisation transfer require attention. The cold-start problem, the inability to build reliable predictions for projects with little historical data, remains a structural limitation of supervised SDP [28], [52], [53]. Transfer learning approaches, including the instance-based and feature-based transfer methods developed in the cross-project defect prediction literature, warrant systematic evaluation in the XAI context to determine whether transferability and interpretability can be maintained simultaneously.

Line-level and commit-level granularity at scale. LINE-DP and JIT prediction frameworks have demonstrated feasibility on open-source Java projects [27], [34], but evaluation at an industrial scale, large proprietary codebases, polyglot projects, and high-velocity commit streams has not been conducted. Scalability analysis and integration studies in real CI/CD environments are necessary before line-level XAI-assisted prediction can be

recommended as a production tool. Lastly, the dataset quality and benchmark need strong reform. The comparability of results between research is frequently hampered by the use of SZZ-derived labels and ageing NASA/PROMISE benchmarks [28], [32], and [53]. The field would benefit from a carefully checked, community-curated benchmark dataset that clearly labels provenance and explicitly accounts for duplicate instances. Additionally, the information should cover microservices, continuous deployment, and multi-language repositories in order to represent contemporary development processes. Radliqué [46] also found that, for some model families, increasing dataset volume may matter more than reducing noise. This finding warrants further investigation into how dataset size and quality should be balanced across different contexts, which could help guide data collection practices.

5. CONCLUSION

Machine learning and deep learning models achieve high prediction accuracy on benchmark SDP datasets, but their industrial use remains constrained when practitioners cannot verify, trust, or act on unexplained predictions. Across the 40 reviewed studies, explainability, data quality, and prediction granularity emerged as the main research concerns. Post-hoc methods, particularly LIME, SHAP, and BreakDown, dominated the literature, although repeated assessments showed changes in contribution scores and feature rankings across configurations, raising concerns about explanation consistency and reliability. Data-quality problems, including multicollinearity, label noise, class imbalance, and class overlap, also often affected prediction and explanation reliability more than model architecture, highlighting the need to consider preprocessing alongside model selection. Interest in Just-In-Time (JIT) and line-level prediction is growing, but their applicability to industrial software with high commit velocity and polyglot codebases remains insufficiently validated. These findings informed AISD-F, a modular pipeline combining data quality assurance, hybrid model selection, dual-layer explanation, and CI/CD integration. It supports resource allocation through stable global explanations and presents local explanations with their limitations, while exposing disagreement between explanation methods rather than masking instability. AISD-F remains conceptual and requires empirical validation, including deployment and evaluation in live CI/CD pipelines, before industrial adoption can be recommended.

A major gap is the lack of human-centred evaluation. None of the 40 studies provided controlled evidence that explainable SDP helps developers resolve defects more quickly or accurately than conventional prediction systems. Claims about improved developer outcomes therefore remain unverified. Future research should evaluate explanation support across developer roles and project settings using measures such as resolution time and accuracy. The review has four main limitations. First, searching five databases may have excluded relevant studies indexed elsewhere. Second, although Google Scholar results were cross-checked against other databases, they are less reproducible than results from curated indices. Third, studies in any human language were eligible, but no non-English screening or translation protocol was used, potentially introducing an English-language skew. Finally, the 40 studies covered related but distinct SDP tasks and prediction granularities, along with a small number of adjacent tasks retained for XAI relevance; therefore, some cross-study comparisons should be interpreted as indicative rather than strictly like-for-like.

ACKNOWLEDGMENT

This research was supported by the Unit for Data Science and Computing, the Department of Computer Science at the North-West University, South Africa.

REFERENCES

- [1]. M. Mustaqeem, M. Alam, S. Mustajab, F. Alshanketi, S. Alam, and M. Shuaib, "Comprehensive Bibliographic Survey and Forward-Looking Recommendations for Software Defect Prediction: Datasets, Validation Methodologies, Prediction Approaches, and Tools," *IEEE Access*, Dec. 2024, doi: 10.1109/ACCESS.2024.3517419.
- [2]. K. Bhandari, K. Kumar, and A. L. Sangal, "Data Quality Issues in Software Fault Prediction: A Systematic Literature Review," *Artif. Intell. Rev.*, vol. 56, pp. 7839–7908, 2023, doi: 10.1007/s10462-022-10371-6.
- [3]. B. Isong and E. Igo, "Ensemble Learning for Software Defect Prediction: Performance, Practicality and Future Directions," *J. Inf. Syst. Inform.*, vol. 7, no. 3, pp. 2245–2291, 2025, doi: 10.51519/journalisi.v7i3.1171.
- [4]. J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A Systematic Literature Review on Software Defect Prediction Using Artificial Intelligence:

- Datasets, Data Validation Methods, Approaches, and Tools," *Eng. Appl. Artif. Intell.*, vol. 111, Art. no. 104773, 2022.
- [5]. T. Hasan and S. M. Mohi-Aldeen, "Software Defect Prediction Based on Deep Learning Algorithms: A Systematic Literature Review," *Al-Rafidain J. Comput. Sci. Math. (RJCSM)*, vol. 19, no. 1, pp. 67–79, Jun. 2025, doi: 10.33899/csmj.2025.156086.1160.
 - [6]. A. H. Mohammadkhani, N. S. Bommi, M. Daboussi, O. Sabnis, C. Tantithamthavorn, and H. Hemmati, "A Systematic Literature Review of Explainable AI for Software Engineering," *Inf. Softw. Technol.*, 2023.
 - [7]. E. S. Ortigossa, T. Gonçalves, and L. G. Nonato, "Explainable Artificial Intelligence (XAI)—From Theory to Methods and Applications," *IEEE Access*, vol. 12, pp. 83661–83733, 2024, doi: 10.1109/ACCESS.2024.3409843.
 - [8]. Gezici and A. K. Tarhan, "Explainable AI for Software Defect Prediction with Gradient Boosting Classifier," in *Proc. 7th Int. Conf. Comput. Sci. Eng. (UBMK)*, Diyarbakir, Turkey, 2022, pp. 1–6, doi: 10.1109/UBMK55850.2022.9919490.
 - [9]. M. A. Awal and C. K. Roy, "EvaluateXAI: A Framework to Evaluate the Reliability and Consistency of Rule-Based XAI Techniques for Software Analytics Tasks," *J. Syst. Softw.*, vol. 217, p. 112159, 2024.
 - [10]. S. Stradowski and L. Madeyski, "Interpretability/Explainability Applied to Machine Learning Software Defect Prediction: An Industrial Perspective," *IEEE Softw.*, Nov. 2024, doi: 10.1109/MS.2024.3505544.
 - [11]. O. R. Polu, "Machine Learning for Predicting Software Project Failure Risks," *Int. J. Comput. Eng. Technol.*, vol. 15, no. 4, pp. 950–959, 2024.
 - [12]. J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models," *IEEE Trans. Softw. Eng.*, Mar. 2020, doi: 10.1109/TSE.2020.2982385.
 - [13]. B. Asal and M. Ö. Demir, "Enhancing Software Defect Prediction Through Explainable AI: Integrating SHAP and LIME in a Voting Classifier Framework," in *Proc. 8th Int. Artif. Intell. Data Process. Symp. (IDAP)*, Malatya, Turkiye, 2024, pp. 1–7, doi: 10.1109/IDAP64064.2024.10710700.
 - [14]. A. Khan, A. Ali, J. Khan, F. Ullah, and M. A. Khan, "A Systematic Literature Review of Explainable Artificial Intelligence (XAI) in Software Engineering (SE)," *Research Square*, Sep. 8, 2023, doi: 10.21203/rs.3.rs-3209115/v1.

- [15]. M. Mersha, K. Lam, J. Wood, A. AlShami, and J. Kalita, "Explainable Artificial Intelligence: A Survey of Needs, Techniques, Applications, and Future Direction," *Neurocomputing*, vol. 599, Art. no. 128111, 2024, doi: 10.1016/j.neucom.2024.128111.
- [16]. J. Shin, R. Aleithan, J. Nam, J. Wang, N. S. Harzevili, and S. Wang, "An Empirical Study on the Stability of Explainable Software Defect Prediction," in *Proc. 30th Asia-Pacific Softw. Eng. Conf. (APSEC)*, Seoul, Republic of Korea, 2023, pp. 141–150, doi: 10.1109/APSEC60848.2023.00024.
- [17]. G. Xu, C. Feng, X. Guo, Z. Zhu, and W. Wang, "A Joint Learning Framework for Bridging Defect Prediction and Interpretation," *IEEE Trans. Rel.*, 2025, doi: 10.1109/TR.2025.3586671.
- [18]. Gezici Geçer and A. Kolukisa Tarhan, "Explainable AI Framework for Software Defect Prediction," *J. Softw.: Evol. Process*, vol. 37, no. 4, p. e70018, 2025.
- [19]. G. Esteves, E. Figueiredo, A. Veloso, *et al.*, "Understanding Machine Learning Software Defect Predictions," *Autom. Softw. Eng.*, vol. 27, pp. 369–392, 2020.
- [20]. L. Arora, S. S. Girija, S. Kapoor, A. Raj, D. Pradhan and A. Shetgaonkar, "Explainable Artificial Intelligence Techniques for Software Development Lifecycle: A Phase-specific Survey," in *Proc. 2025 IEEE 49th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Toronto, ON, Canada, 2025, pp. 2281–2288, doi: 10.1109/COMPSAC65507.2025.00321.
- [21]. G. Lee and S. U. -J. Lee, "An Empirical Comparison of Model-Agnostic Techniques for Defect Prediction Models," in *Proc. 2023 IEEE Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Taipa, Macao, 2023, pp. 179–189, doi: 10.1109/SANER56733.2023.00026.
- [22]. R. Jia, "A Brief Analysis of the Progress and Trends in Software Defect Prediction Methods," in *Proc. 5th Int. Conf. Signal Process. Mach. Learn.*, 2025, doi: 10.54254/2755-2721/99/20251769.
- [23]. Q. Song, Y. Guo, and M. Shepperd, "A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1253–1269, Dec. 2019, doi: 10.1109/TSE.2018.2836442.
- [24]. L. Gong, H. Zhang, J. Zhang, M. Wei, and Z. Huang, "A Comprehensive Investigation of the Impact of Class Overlap on Software Defect Prediction," *IEEE Trans. Softw. Eng.*, Apr. 2023, doi: 10.1109/TSE.2022.3220740.
- [25]. I. Mehmood, S. Shahid, H. Hussain, I. Khan, S. Ahmad, S. Rahman, N. Ullah, and S. Huda, "A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine

- Learning," *IEEE Access*, vol. 11, pp. 60710–60726, 2023, doi: 10.1109/ACCESS.2023.3287326.
- [26]. S. Cao, X. Sun, R. Widyasari, D. Lo, X. Wu, L. Bo, J. Zhang, B. Li, W. Liu, D. Wu, and Y. Chen, "A Systematic Literature Review on Explainability for Machine/Deep Learning-Based Software Engineering Research," *arXiv*, arXiv:2401.14617, Jan. 2024.
- [27]. S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, and K. Matsumoto, "Predicting Defective Lines Using a Model-Agnostic Technique," *IEEE Trans. Softw. Eng.*, Sep. 2020, doi: 10.1109/TSE.2020.3023177
- [28]. C. Chai, G. Fan, H. Yu, *et al.*, "Exploring Better Alternatives to Size Metrics for Explainable Software Defect Prediction," *Softw. Qual. J.*, vol. 32, pp. 459–486, 2024.
- [29]. L. Schulte, B. Ledel, and S. Herbold, "Studying the Explanations for the Automated Prediction of Bug and Non-Bug Issues Using LIME and SHAP," *Empir. Softw. Eng.*, vol. 29, no. 93, 2024, doi: 10.1007/s10664-024-10469-1.
- [30]. A. Jude and J. Uddin, "Explainable Software Defects Classification Using SMOTE and Machine Learning," *Ann. Emerg. Technol. Comput. (AETiC)*, vol. 8, no. 1, pp. 36–49, 2024.
- [31]. T. O. Olaleye, D. A. Aborishade, O. Arogundade, A. Abayomi-Alli, and O. J. Adeniran, "Multilayer Perceptron of Software Complexity Metrics for Explainable Multicollinearity Mitigation and Defect Localisation," *Cureus J. Comput. Sci.*, vol. 2, no. 1, 2025, doi: 10.7759/s44389-024-02871-z.
- [32]. X. Wu, W. Zheng, X. Xia, and D. Lo, "Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction," *IEEE Trans. Softw. Eng.*, Jul. 2022, doi: 10.1109/TSE.2021.3063727.
- [33]. R. Malhotra and K. Khan, "OpTunedSMOTE: A Novel Model for Automated Hyperparameter Tuning of SMOTE in Software Defect Prediction," *Intell. Data Anal.*, vol. 29, no. 3, pp. 787–807, 2025, doi: 10.1177/1088467X241301390.
- [34]. R. He, Y. Li, and C. Sun, "Understanding Software Defect Prediction Through eXplainable Neural Additive Models," *IEEE Access*, vol. 13, pp. 82860–82873, 2025, doi: 10.1109/ACCESS.2025.3567140.
- [35]. S. Kabir, M. S. Hossain, and K. Andersson, "A Review of Explainable Artificial Intelligence from the Perspectives of Challenges and Opportunities," *Algorithms*, vol. 18, no. 9, Art. no. 556, Sep. 2025, doi: 10.3390/a18090556
- [36]. V. Chamola, V. Hassija, A. R. Sulthana, D. Ghosh, D. Dhingra, and B. Sikdar, "A Review of Trustworthy and Explainable Artificial Intelligence (XAI)," *IEEE Access*, vol. 11, pp. 78994–79012, 2023, doi: 10.1109/ACCESS.2023.3294569

- [37]. M. J. Page et al., "PRISMA 2020 Explanation and Elaboration: Updated Guidance and Exemplars for Reporting Systematic Reviews," *BMJ*, vol. 372, p. n160, 2021, doi: 10.1136/bmj. n160.
- [38]. M. Begum *et al*., "LCNN: Lightweight CNN Architecture for Software Defect Feature Identification Using Explainable AI," *IEEE Access*, vol. 12, pp. 55744–55756, 2024, doi: 10.1109/ACCESS.2024.3388489.
- [39]. J. Humphreys and H. K. Dam, "An Explainable Deep Model for Defect Prediction," in *Proc. IEEE/ACM 7th Int. Workshop Realising Artificial Intelligence Synergies in Software Engineering (RAISE)*, Montreal, QC, Canada, 2019, pp. 49–55, doi: 10.1109/RAISE.2019.00016.
- [40]. D. Li, Y. Liu, and J. Huang, "Assessment of Software Vulnerability Contributing Factors by Model-Agnostic Explainable AI," *Mach. Learn. Knowl. Extr.*, vol. 6, no. 2, pp. 1087–1113, May 2024, doi: 10.3390/make6020050.
- [41]. T. Ganz, *Software Defect Localization Using Explainable Deep Learning*, Ph.D. dissertation, Fakultät IV – Elektrotechnik und Informatik, Technische Universität Berlin, Berlin, Germany, 2024.
- [42]. I. Siachos, N. Kanakaris, and N. Karacapilidis, "Software Bug Prediction Using Graph Neural Networks and Graph-Based Text Representations," *Expert Syst. Appl.*, vol. 259, Art. no. 125290, 2025, doi: 10.1016/j.eswa.2024.125290.
- [43]. N. S. Bommi and A. Negi, "A Standard Baseline for Software Defect Prediction: Using Machine Learning and Explainable AI," in *Proc. IEEE 47th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Torino, Italy, 2023, pp. 1798–1803, doi: 10.1109/COMPSAC57700.2023.00278.
- [44]. M. Mustaqeem, S. Mustajab, M. Alam, F. Jeribi, S. Alam, and M. Shuaib, "A Trustworthy Hybrid Model for Transparent Software Defect Prediction: SPAM-XAI," *PLoS ONE*, vol. 19, no. 7, Art. no. e0307112, Jul. 2024, doi: 10.1371/journal.pone.0307112.
- [45]. A. P. Meyer, A. Albarghouthi, and L. D'Antoni, "The Dataset Multiplicity Problem: How Unreliable Data Impacts Predictions," in *Proc. 2023 ACM Conf. Fairness, Accountability, Transpar. (FAccT)*, Chicago, IL, USA, Jun. 2023, pp. 320–331, doi: 10.1145/3593013.3593988.
- [46]. Ł. Radliński, "The Impact of Data Quality on Software Testing Effort Prediction," *Electronics*, vol. 12, no. 7, Art. no. 1656, Mar. 2023, doi: 10.3390/electronics12071656.

- [47]. T. N. Al-Hadidi and S. O. Hasoon, "Software Defect Prediction Using Extreme Gradient Boosting (XGBoost) with Optimisation Hyperparameter," *Al-Rafidain J. Comput. Sci. Math.*, Jan. 2024, doi: 10.33899/CSMJ.2023.142739.108.
- [48]. D. Al-Fraihat, Y. Sharrab, A.-R. Al-Ghuwairi, H. Alshishani, and A. Algarni, "Hyperparameter Optimisation For Software Bug Prediction Using Ensemble Learning," *IEEE Access*, Mar. 2024, doi: 10.1109/ACCESS.2024.3380024.
- [49]. M. Das, B. R. Mohan, R. M. R. Guddeti, and N. Prasad, "Hybrid Bio-Optimized Algorithms for Hyperparameter Tuning in Machine Learning Models: A Software Defect Prediction Case Study," *Mathematics*, vol. 12, no. 16, p. 2521, Aug. 2024, doi: 10.3390/math12162521.
- [50]. I. D. Raji, H. Bello-Salau, I. J. Umoh, A. J. Onumanyi, M. A. Adegboye, and A. T. Salawudeen, "Simple Deterministic Selection-Based Genetic Algorithm for Hyperparameter Tuning of Machine Learning Models," *Appl. Sci.*, vol. 12, no. 3, Art. no. 1186, Jan. 2022, doi: 10.3390/app12031186.
- [51]. C. Antal-Vaida, "Basic Hyperparameters Tuning Methods for Classification Algorithms," *Inform. Econ.*, vol. 25, no. 2, pp. 64–74, 2021, doi: 10.24818/issn14531305/25.2.2021.06.
- [52]. S. Haldar and L. F. Capretz, "Explainable Software Defect Prediction from Cross-Company Project Metrics Using Machine Learning," in *Proc. 7th Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Madurai, India, 2023, pp. 150–157, doi: 10.1109/ICICCS56967.2023.10142534.
- [53]. L. Madeyski and S. Stradowski, "Predicting Test Failures Induced by Software Defects: A Lightweight Alternative to Software Defect Prediction and Its Industrial Application," *J. Syst. Softw.*, vol. 222, Art. no. 112360, 2025, doi: 10.1016/j.jss.2024.112360.
- [54]. P. Tadapaneni, N. C. Nadella, M. Divyanjali, and Y. Sangeetha, "Software Defect Prediction Based on Machine Learning and Deep Learning," *Proc. 2022 Int. Conf. Inventive Comput. Technol. (ICICT)*, Nepal, 2022, pp. 116–122, doi: 10.1109/ICICT54344.2022.9850643.
- [55]. W. Albattah and M. Alzahrani, "Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach," *AI*, vol. 5, no. 4, pp. 1743–1758, Sep. 2024, doi: 10.3390/ai5040086.
- [56]. L. Chmielowski, M. Kucharzak, and R. Burduk, "Application of Explainable Artificial Intelligence in Software Bug Classification," *IAPGOS*, no. 1, pp. 14–17, 2023, doi: 10.35784/iapgos.3396.

- [57]. M. Begum, M. H. Shuvo, I. Ashraf, A. A. Mamun, J. Uddin, and M. A. Samad, "Software Defects Identification: Results Using Machine Learning and Explainable Artificial Intelligence Techniques," *IEEE Access*, vol. 11, pp. 132750–132765, 2023.