



ARTICLE

Integrating Explainable Artificial Intelligence into Software Defect Prediction: Addressing the Black Box Challenge

Miriam Ramírez-Zárate^{ID}, Ángel J. Sánchez-García^{*ID}, Lizbeth A. Hernández-González^{ID} and Saúl Domínguez-Isidro^{ID}

Facultad de Estadística e Informática, Universidad Veracruzana, Xalapa, Veracruz, Mexico

*Corresponding Author: Ángel J. Sánchez-García. Email: angesanchez@uv.mx

Received: 02 May 2026; Accepted: 28 August 2026; Published: 22 October 2026

ABSTRACT: Software Defect Prediction (SDP) plays a critical role in improving software quality by identifying fault-prone modules early in the development lifecycle. While machine learning models (particularly black-box) have demonstrated high predictive accuracy, their lack of interpretability limits their adoption in environments where transparency and trust are essential. In contrast, naturally explainable models such as Logistic Regression, Decision Trees, and Naïve Bayes offer intrinsic interpretability but may underperform in complex defect prediction tasks. This paper presents a comparative empirical evaluation of seven classification algorithms (three naturally explainable and four black-box) across five datasets from the NASA PROMISE repository. Black-box models are enhanced with post-hoc Explainable Artificial Intelligence (XAI) techniques, namely LIME, SHAP, and Breakdown. To assess the quality of the generated explanations, a set of interpretability metrics including fidelity, stability, complexity, and a composite interpretability score is proposed. Our results show that black-box models achieved superior predictive performance in one out of five datasets (JM1), while in the remaining datasets, naturally explainable models were either competitive or superior. Notably, Logistic Regression achieved the highest F1-scores in KC1, KC2, and PC1, surpassing all black-box models in those datasets, and tied with SVM in CM1. Among XAI techniques, LIME offers the most concise explanations (complexity = 0.476), while SHAP and Breakdown provide complete coverage (complexity = 1.0). Random Forest and Gradient Boosting achieve the highest composite interpretability scores (0.92), indicating reliable and consistent explanations. Our findings suggest that post-hoc XAI techniques are justified when black-box models provide clear predictive advantages, but naturally explainable models should not be overlooked, particularly when interpretability is a primary requirement.

KEYWORDS: Explainable Artificial Intelligence (XAI); software defect prediction (SDP); black-box models; post-hoc explanations; interpretability metrics

1 Introduction

The development and maintenance of software constitute one of the principal tasks of software engineering (SE), a discipline that employs principles aimed at ensuring not only that programs function correctly, but also that they are efficient and secure. According to Pressman [1], software engineering is the application of a systematic, disciplined, and quantifiable approach to the development, operation, and maintenance of software. This approach is relevant not only to developers but also to other professionals involved in the software development life cycle, such as managers, testers, requirements analysts, and other software engineering practitioners.

A quality software product is one that meets user-established requirements, is free of critical defects, and is delivered within agreed-upon deadlines and budgets. However, building defect-free software remains a persistent challenge, as defects can arise at any stage of the software development life cycle—from requirements to implementation and testing. These defects not only affect software performance but also increase correction costs and may reduce user confidence in the product [1,2]. Software quality is a critical factor for the success of any project, since undetected defects can lead to increased costs, delivery delays, and a decrease in user trust [1,3].

According to a study by the IEEE Computer Society [3], defects detected in late development stages can increase correction costs by up to 100 times compared to those identified during the design phase.

Due to the negative impact of defects, early defect identification is a priority in software engineering. Early defect detection is crucial for reducing correction-related costs and improving software quality [2]. This need has driven the development of advanced techniques for Software Defect Prediction (SDP), which aims to help detect faults before they emerge in advanced stages of the development life cycle. SDP is based on the analysis of historical data, such as code metrics and previous defect records, thereby enabling development teams to prioritize testing and correction efforts on fault-prone modules. Moreover, this approach not only optimizes available resources but also significantly enhances software quality [2].

To make defect prediction useful, researchers have opted to implement classification algorithms based on machine learning (ML), which analyze large volumes of data to identify patterns associated with defects. However, according to [2], the effectiveness of these algorithms depends on model selection and the quality of the data used. Consequently, several studies have attempted to evaluate the performance of different algorithms in the SDP context.

In addition to classical algorithms, ensemble classification methods have gained popularity in recent years. According to Olivares-Galindo et al. [4], ensemble algorithms can match or exceed the performance of classical algorithms in software defect prediction. This is particularly relevant when working with imbalanced datasets, a common issue in SDP, where most code is free of severe errors. As Dietterich points out [5], ensemble methods are especially useful in such scenarios, as they help balance predictions more effectively and reduce the risk of model bias.

Although SDP has emerged as a key strategy for predicting faults, a persistent challenge is the lack of explainability of the models used. On the one hand, Menzies et al. [2] argue that ML algorithms such as Random Forest and Support Vector Machines (SVM) have demonstrated high levels of accuracy in defect detection. On the other hand, authors such as [6] note that these models often function as “black boxes,” hindering their adoption in environments where transparency and explainability are essential. This lack of interpretability not only reduces developers’ confidence in the results but also limits the identification of the most influential variables in predictions, particularly for software quality improvement.

In response to this problem, advances have emerged in the field of Explainable Artificial Intelligence (XAI). However, despite the growing interest in XAI, its integration into SDP remains in its infancy. Existing studies have largely focused on applying post-hoc explainability techniques, such as SHAP and LIME, to black-box models as an afterthought, without systematically evaluating whether the explanations provided are actually useful for software practitioners [6,7]. This ad-hoc approach presents three critical gaps that remain unaddressed in the current literature.

First, there is no standardized framework for measuring interpretability in the SDP context. While fidelity and stability are common metrics in the broader XAI literature, their adaptation to defect prediction,

where developers need to understand which code metrics (e.g., complexity, coupling, cohesion) drive predictions, has not been systematically validated. Most studies simply report feature importance rankings without assessing whether these rankings are stable across datasets or consistently reflect domain knowledge [4].

Second, the trade-off between predictive accuracy and interpretability has been assumed but never rigorously quantified for SDP. While inherently interpretable models (e.g., decision trees, logistic regression) are often assumed to sacrifice performance compared to black-box ensembles (e.g., Random Forest, XGBoost), this assumption has not been empirically tested using controlled experiments and statistical significance tests. Consequently, practitioners lack evidence-based guidance on when to prioritize interpretability over accuracy, and vice versa.

Third, existing studies have not compared multiple XAI techniques (SHAP, LIME, and Breakdown) side-by-side using the same experimental setup and datasets. This fragmentation prevents the research community from reaching consensus on which technique provides the most reliable and actionable explanations for software defect prediction tasks. As a result, developers and project managers are left with conflicting recommendations and no clear criteria for selecting an appropriate explanation method.

These gaps are not merely academic. In industrial settings, where software quality directly impacts operational costs, user trust, and project timelines, the inability to interpret defect predictions undermines the adoption of ML-based quality assurance tools [3]. Development teams remain reluctant to act on predictions they cannot understand, leading to underutilization of potentially valuable SDP systems.

In industrial settings, software defect prediction has been adopted to optimize testing resources and improve quality assurance processes. For instance, Kim et al. [8] demonstrated that defect prediction models can effectively prioritize code inspections in large-scale industrial projects, reducing the effort required for manual code reviews. Similarly, Jiang et al. [9] reported that personalized defect prediction, tailored to individual developer characteristics, has been successfully deployed in commercial software companies to guide testing efforts. More recently, Wang et al. [10] conducted an empirical study on continuous software bug prediction, analyzing 120 continuously released versions across six large-scale open-source systems, showing the practical value of prediction models in real-world continuous software development scenarios. These real-world applications underscore the practical value of SDP and motivate the need for further improvements in model interpretability, as practitioners require not only accurate predictions but also actionable insights into the factors driving defects.

To address these gaps, this study presents a systematic integration of XAI into the SDP pipeline. Unlike prior work that applies XAI techniques in isolation, it is proposed a comprehensive framework that:

1. evaluates multiple inherently interpretable models alongside black-box models enhanced with SHAP, LIME, and Breakdown;
2. introduces and empirically validates a set of interpretability metrics specifically tailored to SDP, including fidelity (consistency between original and explanatory model predictions) and stability (robustness of explanations across cross-validation folds); and
3. quantifies the accuracy-interpretability trade-off using rigorous statistical testing to ensure the reliability of our conclusions.

The potential of ML-based SDP to improve software quality and reduce maintenance costs has been documented in the literature. Herbold [11] developed a cost model for defect prediction, demonstrating that SDP can guide quality assurance resources and generate positive economic returns under realistic conditions, particularly when the relationship between artifacts and defects is properly accounted for. Additionally, the systematic literature review by Hall et al. [12] found that accurate fault prediction can help direct test effort and reduce overall project costs. However, it is important to clarify that cost reduction analysis is not

part of the research contributions of this study. Our work focuses on the integration of XAI with SDP to improve interpretability, which we argue is a prerequisite for the adoption of SDP in practice. We do not conduct a direct economic analysis of cost savings, as this would require a separate study involving industrial deployment and cost-benefit analysis.

Therefore, the combination of ML and XAI techniques presents a promising approach to improving software quality and reducing defect-related costs, thereby fostering more robust and efficient software development.

The principal objective of this work is to bridge the identified research gaps by systematically evaluating classification algorithms within an XAI-enhanced SDP framework. Specifically, the predictive performance and interpretability of inherently explainable models (decision trees, logistic regression, and Naïve Bayes) is compared against black-box models (Random Forest, SVM, XGBoost) augmented with post-hoc explanation techniques: SHAP, LIME, and Breakdown. Our goal is to provide empirical evidence on whether the accuracy-interpretability trade-off is justified and to offer practical guidance for practitioners selecting SDP models.

Our main contributions are as follows:

1. **Systematic integration of XAI into SDP.** A comprehensive framework that embeds explainability as a first-class concern in the defect prediction pipeline, rather than as an afterthought is proposed.
2. **Novel interpretability metrics for SDP.** Fidelity and stability metrics specifically designed to assess the quality of explanations in the context of software defect prediction are defined and empirically validated, addressing the current lack of standardized evaluation criteria.
3. **Comparative evaluation of multiple XAI techniques.** The first side-by-side comparison of SHAP, LIME, and Breakdown within the same experimental framework is provided, enabling direct assessment of their relative strengths and weaknesses for SDP.

Finally, research workflow diagram [Fig. 1](#) illustrating the main stages of the study, grouping activities into problem definition, literature review, experimental design, explainability analysis and conclusions and future work.

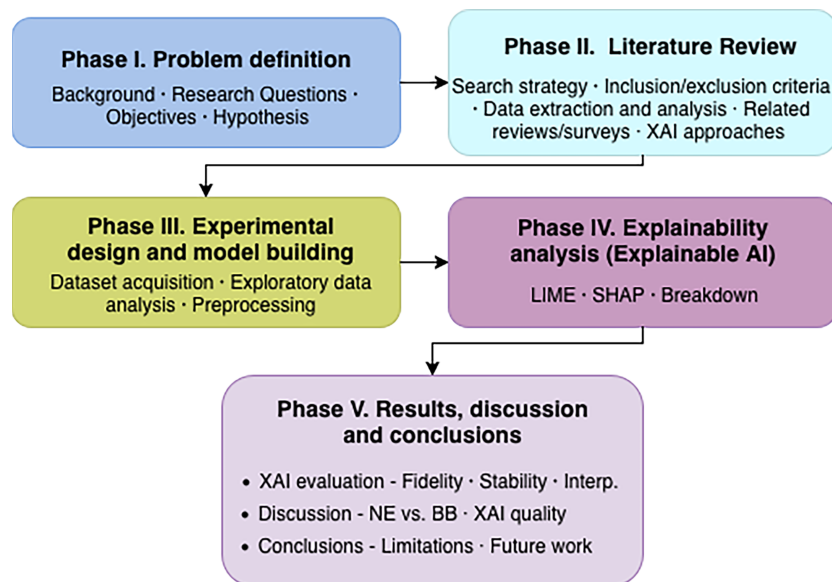


Figure 1: Research workflow diagram illustrating the main stages of the study.

The remainder of this article is structured as follows. [Section 2](#) reviews related work on software defect prediction and XAI techniques. [Section 3](#) describes the research method, including datasets, preprocessing, and post-hoc methods. [Section 4](#) details the empirical study design. [Section 5](#) explains model evaluation metrics. [Section 6](#) presents the experimental results in terms of predictive performance and explainability metrics. [Section 7](#) discusses the findings, compares the XAI techniques, and examines the trade-offs between accuracy and interpretability. [Section 8](#) discusses threats to validity. Finally, [Section 9](#) presents conclusions and outlines directions for future work.

2 Related Work

The literature on SDP contains several Systematic Literature Reviews (SLRs), each focusing on different research questions and prediction methodologies. As shown in [Table 1](#), although a considerable number of secondary studies exist within the SDP domain, none of these systematic reviews specifically examine XAI algorithms as their primary focus. However, this does not mean that XAI has been entirely unexplored. [Tables 2–4](#) summarize recent primary studies that employ LIME, SHAP, and Breakdown.

Table 1: Secondary studies related to software defect prediction. NS (Number of studies).

Study	Year	Years Covered	NS	ML Algorithms
[13]	2021	2016–2020	38	Machine learning algorithms in general
[14]	2021	2012–2021	46	Algorithms based on ensemble classifiers
[15]	2022	2010–2021	283	Artificial Intelligence approaches
[16]	2022	2015–2018	8	Neural networks
[4]	2023	2017–2022	42	Algorithms based on ensemble classifiers
[17]	2023	2009–2022	52	Ensemble and deep learning techniques
[18]	2023	2018–2023	32	Machine learning algorithms in general

Table 2: Post-hoc LIME algorithm approaches in the last 5 years.

XAI Approach	Black Box Algorithms	Study
LIME	Gradient boosting	[19]
LIME	Random Forest, SVM	[20]
LIME	Voting classifier	[21]
LIME	XGBoost	[22]
LIME	Logistic Regression, Decision Tree, Random Forest, AVNNet, Gradient Boosting, XGBoost	[23]
LIME	SAINT	[24]
LIME	Lightweight CNN	[25]
LIME	Random Forest, Gradient Boosting, XGBoost Tree, MLP	[26]
LIME	Random Forest, XGBoost, SVM	[27]

Table 3: Post-hoc SHAP algorithm approaches in the last 5 years.

XAI Approach	Black Box Algorithms	Study
SHAP	Random Forest, AdaBoost, Gradient Boosting, Extra Tree, XGBoost, CatBoost	[28]
SHAP	Gradient Boosting	[19]
SHAP	Voting classifier	[21]
SHAP	XGBoost	[22]
SHAP	Linear Regression, MLP, Random Forest, Decision Tree, SVM	[29]
SHAP	Lightweight CNN	[25]
SHAP	Random Forest, Gradient Boosting, XGBoost Tree, MLP	[26]
SHAP	RS-XGBoost	[30]
SHAP	LightGBM	[31]
SHAP	XG-HARSA	[32]
SHAP	CatBoost	[33]
SHAP	Logistic Regression, KNN, Decision Tree, Random Forest, SVM, AdaBoost, Gradient Boosting, SGB, XGBoost, CatBoost, Stacking	[34]

Table 4: Post-hoc BreakDown algorithm approaches in the last 5 years.

XAI Approach	Black Box Algorithms	Study
BreakDown	Logistic Regression, Decision Tree, Random Forest, AVNNet, Gradient Boosting, XGBoost	[23]
BreakDown	Decision Tree	[35]
BreakDown	Random Forest, Gradient Boosting, XGBoost Tree, MLP	[26]

From the set of studies reviewed in [Table 1](#), together with additional XAI-focused research in the SDP domain, the findings are summarized in [Tables 2–4](#).

To contextualize our work within the existing literature, we conducted a systematic review of SDP and XAI research. [Table 1](#) presents secondary studies (systematic reviews and mapping studies) published between 2021 and 2023, covering classical ML, ensemble, and deep learning approaches. Notably, most of these studies do not focus on interpretability, underscoring the relevance of our XAI-focused investigation.

[Tables 2–4](#) summarize primary studies applying LIME, SHAP, and Breakdown to SDP over the last five years. [Table 2](#) shows that LIME has been applied primarily to tree-based ensembles (Random Forest, XGBoost) and, more recently, to lightweight neural networks. [Table 3](#) reveals that SHAP is the most widely adopted technique, applied to the broadest range of models, including classical algorithms, tree-based ensembles, and advanced variants such as RS-XGBoost and stacking ensembles. Its theoretical foundations and desirable properties (consistency and fairness) make it particularly suitable for rigorous explanation tasks. [Table 4](#) shows that Breakdown has been less frequently applied, primarily to tree-based ensembles and classical models, though its computational efficiency makes it a viable alternative for resource-constrained environments.

Recent advances in software defect prediction have explored the integration of metaheuristic optimization techniques with machine learning algorithms to enhance predictive performance. For instance, Ref. [36] proposed an AdaBoost-based approach optimized using the Sinh Cosh optimizer, demonstrating

that metaheuristic tuning can significantly improve the detection of defective modules compared to baseline AdaBoost configurations. Similarly, Ref. [37] applied a modified metaheuristic algorithm to tune machine learning hyperparameters for defect detection, reporting substantial improvements in F1-score and recall across multiple PROMISE datasets. These studies highlight the growing interest in hybrid optimization-ML approaches for SDP, complementing the XAI-focused framework by addressing the performance side of the accuracy-interpretability trade-off.

While this work does not employ metaheuristic optimization, our systematic comparison of naturally explainable and black-box models, combined with post-hoc XAI techniques, provides a complementary perspective by emphasizing interpretability alongside predictive performance.

3 Research Method

3.1 Data Sets

SDP is a fundamental aspect of software development and maintenance processes, as it is closely related to the overall success of the software product [38].

The PROMISE repository [39] is one of the most widely used sources of software defect prediction data in the research community. It was selected for this study for several reasons. First, it provides standardized, publicly available datasets that enable comparability with prior studies, which is essential for benchmarking new approaches [2,40]. Second, the repository includes multiple datasets with varying characteristics (e.g., size, language, domain), allowing for a diverse evaluation of our framework. Third, the datasets are well-documented and have been extensively used in SDP literature, which facilitates replication and validation of our results.

The datasets used for the research in the software defect prediction area are: CM1, JM1, KC1, KC2, and PC1. Each dataset has a title, an original source with the donor's contact information and respective date, a brief description of usage and results, the language in which it is written, the number of instances, the number of attributes, and the distribution of the Defect/Problems class [39]. Each of the datasets used is donated by NASA and has a different number of instances, as is shown in Table 5. However, each dataset records the same 22 information attributes (five different lines of code measure, three McCabe metrics, four base Halstead measures, eight derived Halstead measures, one branch count, and one target field), as shown in Table 6.

Table 5: Number of instances for each dataset.

Data Set	Number of Instances
CM1	498
JM1	10,885
KC1	2109
KC2	522
PC1	1109

Table 6: Attribute information.

Attribute	Overview
Loc	M McCabe's line count of code
v(g)	M McCabe "cyclomatic complexity"

(Continued)

Table 6 (continued)

Attribute	Overview
ev(g)	McCabe “essential complexity”
iv(g)	McCabe “design complexity”
N	Halstead total operators + operands
V	Halstead “volume”
L	Halstead “program length”
d	Halstead “difficulty”
i	Halstead “intelligence”
e	Halstead “effort”
b	Halstead
t	Halstead’s time estimator
lOcode	Halstead’s line count
lOComment	Halstead’s count of lines of comments
lOBlank	Halstead’s count of blank lines
lOCode And Comment	Line of Code and Comment Count
uniqOp	Unique operators
uniqOpnd	Unique operands
totalOp	Total operators
totalOpnd	Total operands
branchCount	Numeric of the flow graph
defects/problems	False, true module has/has not one or more reported defects.

3.2 Dataset Preprocessing

Once the exploratory analysis was completed, data preprocessing was carried out with the aim of preparing the data for training the prediction models.

Preprocessing is a critical phase within the workflow, as it addresses issues such as missing values, duplicate data, differences in scale among variables, and the presence of irrelevant attributes. These conditions can negatively affect the performance and stability of the models.

In this study, the preprocessing process included data cleaning, handling of missing values, removal of irrelevant attributes, and normalization of numerical variables.

This process was applied uniformly across all datasets using a preparation pipeline implemented in Python, leveraging the pandas and scikit-learn libraries.

3.2.1 Handling of Missing Values

Missing values, also referred to in the literature as Missing Values (MV), represent a frequent challenge when collecting data in real-world environments. The presence of such values can affect the quality and accuracy of prediction models; therefore, it is necessary to apply appropriate strategies for their treatment [41].

Among the most common techniques are the removal of incomplete records and data imputation using substitute values. In this study, given that approximately 40% of the instances in the KC1 dataset contained missing values, we chose to discard the affected data rather than perform imputation. Imputing nearly half of

the dataset would have introduced a substantial proportion of synthetic data, potentially biasing downstream analyses. Table 7 presents the updated number of instances per dataset.

Table 7: Number of instances per dataset after handling missing values.

Dataset	Number of Instances	Update of Number of Instances
CM1	498	443
JM1	10,885	8839
KC1	2109	1274
KC2	522	389
PC1	1109	949

3.3 Separation of Predictor Variables and Target Variable

Once the cleaning process and handling of missing values were completed, the predictor variables (features) were separated from the target variable. In all datasets, the target variable used was `defects`, which indicates in a binary manner whether a software module presents defects or not.

During this stage, the presence of the `project` column in each dataset was also verified. This column, functioning solely as a project identifier, does not provide relevant information for prediction and was therefore excluded from the set of predictor variables. This separation ensures that machine learning models are trained only with features that can contribute to predictive performance, avoiding noise or irrelevant information that could affect model generalization [42].

3.4 Variable Normalization

Finally, a standardization process was applied to the numerical variables using the `StandardScaler` technique, which transforms the data so that each feature has a mean of zero and a standard deviation of one.

Feature scaling is essential in many machine learning algorithms, especially those based on distances or gradients, such as logistic regression, support vector machines, and neural networks [43]. The absence of scaling can cause variables with larger ranges to dominate the learning process, affecting model convergence and performance.

3.5 Pos-Hoc Methods

The selection of LIME, SHAP, and Breakdown as the post-hoc explanation techniques in this study was motivated by three main factors. First, all three methods are model-agnostic, meaning they can be applied to any black-box classifier without requiring access to the model's internal structure [7,44,45]. This property is essential for our comparative analysis, as it allows consistent evaluation across different model types (Random Forest, Gradient Boosting, SVM, and MLP). Second, these techniques represent different paradigms of explanation generation, enabling a comprehensive comparison [6,46]. LIME generates explanations by approximating the local behavior of the model using an interpretable surrogate model, making it suitable for understanding individual predictions [7]. SHAP, based on cooperative game theory, provides theoretically grounded explanations with desirable properties such as consistency and fairness [44]. Breakdown offers a computationally efficient alternative that builds explanations through sequential accumulation of feature contributions [45]. Third, these techniques are among the most widely used and well-documented in the XAI literature, with implementations available in popular Python libraries (e.g., `shap`, `lime`, `dalex`). This facilitates replication and validation of our results. It is important to note that each method has

distinct strengths and limitations; for instance, SHAP provides more robust global insights but incurs higher computational cost, while LIME offers simpler explanations but sacrifices some stability [6,46]. Recent studies in software defect prediction have also employed these techniques to generate visual explanations that help practitioners understand which characteristics are most associated with software defects [26].

3.5.1 Local Interpretable Model-Agnostic Explanations (LIME)

Local Interpretable Model-Agnostic Explanations (LIME) is one of the most influential post-hoc techniques in the field of Explainable Artificial Intelligence. Proposed by [7] at the ACM SIGKDD conference, it addresses the opacity problem of complex models. Its main objective is to explain individual predictions without modifying the original model. As a model-agnostic method, it can be applied to any classifier or regressor, provided its predictions are accessible.

The core idea of LIME is relatively straightforward: although a complex model may be difficult to interpret globally, its behavior around a specific instance can be approximated using a simpler model. In other words, LIME assumes that, within a small region of the data space, even a highly non-linear model can behave approximately linearly.

LIME formulates an optimization problem that seeks a simple, interpretable model g belonging to a family of models, satisfying two conditions: faithfully approximating the original model's prediction f around the instance of interest, and maintaining low complexity to ensure comprehensibility.

The LIME process consists of four main steps: (1) generating synthetic samples through perturbations around the instance of interest; (2) obtaining predictions from the black-box model for these synthetic instances; (3) assigning weights to each sample based on proximity to the original instance, typically using an exponential kernel function; and (4) fitting an interpretable model using the weighted samples, where the resulting coefficients represent the local contribution of each variable to the specific prediction.

A relevant characteristic of LIME is that the generated explanations are inherently local, describing only the logic behind a particular prediction rather than the model's general behavior. Consequently, two similar instances may receive different explanations if the model exhibits non-linear variations in that region. This characteristic provides granularity but may also generate inconsistencies when comparing multiple explanations [46].

Methodologically, LIME maintains explanation simplicity by limiting the complexity of the generating model, typically implemented through L1 regularization (Lasso), which reduces the number of variables included and retains only the most influential ones.

In the context of Software Defect Prediction (SDP), LIME has been primarily applied to explain high-performance models. The predictive model classifies modules as defective or non-defective based on structural and complexity metrics. Without an explanatory mechanism, it is impossible to know which metrics influenced a specific classification. LIME enables the decomposition of predictions and quantifies the local contribution of metrics such as cyclomatic complexity, coupling, lines of code, or cohesion.

Limitations identified in the literature include variability in explanations across executions due to dependence on random sampling for perturbation generation [7]. Similarly, explanations may vary depending on how "proximity" between instances is defined, as the specific kernel function and region size influence the final result.

3.5.2 Shapley Additive Explanations (SHAP)

Shapley Additive Explanations (SHAP) is one of the most robust methodologies in Explainable Artificial Intelligence. Proposed by [44], it is based on cooperative game theory developed by Lloyd Shapley in 1953. The central idea is to transfer the concept of “fair distribution” of gains among players to the problem of explaining predictions in machine learning models.

In this approach, each feature of the model is considered a “player” participating in generating the final prediction. Just as cooperative game theory determines each player’s contribution to the total outcome, SHAP calculates how much each input variable contributes to the model’s predicted value.

The mathematical foundation comes from Shapley values, which offer a fair and well-defined way to distribute contributions. To calculate this contribution, the effect of each feature when incorporated into different subsets of variables is analyzed. Instead of relying on a single order of incorporation, SHAP averages the marginal effect of each variable over all possible combinations, yielding more stable and theoretically justified feature importance measures.

SHAP satisfies several desirable mathematical properties not always present in other post-hoc techniques. These include efficiency (the sum of all individual contributions equals the difference between the prediction and the model’s base value), symmetry (two variables with identical impact receive identical attribution), dummy (a feature that does not influence the prediction receives zero value), and consistency (if a variable’s marginal effect increases in a modified model, its SHAP value cannot decrease). These properties provide a solid theoretical foundation distinguishing it from methods based solely on heuristic approximations [44].

In practice, SHAP enables both local and global analyses. Locally, it explains why a specific instance was classified as defective or non-defective. Globally, by aggregating SHAP values across multiple instances, it is possible to identify which variables have the greatest overall influence on the model. Common visualization tools include summary plots, force plots, and dependence plots.

In SDP, SHAP has been applied to interpret complex models such as Random Forest, XGBoost, and Gradient Boosting on datasets like PC1 and others from the NASA PROMISE repository. Its use enables identification of which structural software metrics increase or decrease the probability of defects. However, exact Shapley value calculation can be computationally expensive, prompting the development of efficient variants such as TreeSHAP for tree-based models [47].

3.5.3 Breakdown

Breakdown is a post-hoc explanation technique designed to decompose a model’s prediction into contributions attributable to each feature sequentially. Developed by [45] as part of the interpretability methodological framework later implemented in tools such as the DALEX package in R [48], Breakdown analyzes how a model’s prediction changes as variables are progressively incorporated.

Unlike methods such as SHAP, which are based on cooperative game theory and consider multiple possible variable combinations, Breakdown employs a sequential approach. It starts from a baseline value and adds features one by one according to a defined order. The observed change in prediction after incorporating each variable is interpreted as its individual contribution, generating a trajectory of accumulating contributions and presenting the evolution of the prediction as variables are added.

For a predictive model $f(x)$, a Breakdown explanation can be represented as an additive decomposition where the final prediction is obtained by summing the baseline value and the incremental contributions of each variable according to the established order. Notably, because the result depends on the inclusion

order, different sequences may yield different decompositions, meaning Breakdown does not guarantee a unique explanation.

An advantage is its computational efficiency. By not requiring evaluation of all possible variable combinations, it is more scalable for datasets with many features or models with high evaluation costs.

In SDP, Breakdown allows examination of how structural software metrics incrementally influence the probability of a module being classified as defective. It is commonly visualized using waterfall plots, where the accumulation of contributions can be observed leading to the final prediction.

4 Empirical Study Design

4.1 Dataset Splitting

To objectively evaluate model performance, five-fold stratified cross-validation was employed on each dataset. In this scheme, the complete dataset is divided into five partitions of similar size; in each iteration, four partitions are used for model training and the fifth is reserved for evaluation. After completing the five iterations, all data have participated in both training and testing. This procedure yields a more robust estimate that is less dependent on a particular data split, which is especially relevant for small datasets such as CM1 and KC2 [49].

To ensure that each fold was representative, stratification was applied to the target variable across all partitions, guaranteeing that the proportion of defective and non-defective modules remained consistent across folds. Due to the class imbalance present in all datasets, the SMOTE technique was applied within each fold, exclusively to the training data of that iteration.

Finally, a fixed random seed was used in all experiments, ensuring that the partitions are reproducible in each execution and that any differences in model performance are due to their own characteristics and not to random variations in the data split.

4.2 Class Balancing with SMOTE

As identified during the exploratory analysis, the datasets exhibit class imbalance, where modules without defects represent the majority of instances. This imbalance can negatively affect model training, as classifiers tend to bias toward the majority class when optimizing global metrics such as accuracy, largely ignoring the minority class, which in this case corresponds to defective modules [50].

To address the class imbalance problem identified in the datasets, the SMOTE (Synthetic Minority Over-sampling Technique) technique was used, proposed by Chawla et al. [50]. This technique is designed to improve learning in problems where one class has significantly fewer instances than the other.

Unlike traditional oversampling, which consists of duplicating existing examples of the minority class, SMOTE generates new synthetic instances. To do this, it works in the feature space: for each sample of the minority class, it selects one of its nearest neighbors and creates a new point at an intermediate position between them. In this way, it not only increases the amount of data but also introduces variability, allowing for a better representation of the minority class distribution.

The use of SMOTE contributes to improving model performance, as it avoids problems such as overfitting that usually occur when data is simply duplicated. Furthermore, by generating new, non-identical examples, it helps the model learn broader and more balanced decision boundaries, rather than focusing solely on very specific regions.

It is important to note that SMOTE was applied only to the training set, after data splitting. This decision is critical to avoid the problem known as data leakage, which occurs when information from the test set influences model training.

4.3 Selected Machine Learning Models

In the area of software defect prediction, classical machine learning algorithms have been used as baseline techniques for modeling the relationship between software metrics and the presence of defects. These methods have served as benchmark points in numerous empirical studies on defect prediction [2,40].

Regarding the Section 2, which corresponds to the state of the art in software defect prediction, two classifications of Explainable AI algorithms were identified: naturally explainable algorithms, which correspond to classical machine learning algorithms, and post-hoc methods, which are implemented after the modeling of classical machine learning algorithms that exhibit black-box behavior. The algorithms used based on the findings from the related work and employed in this study are described below.

4.3.1 Naturally Explainable Models

Naturally explainable algorithms are considered interpretable; however, interpretability does not have a single definition. According to [51], interpretability becomes important when what the model optimizes (such as minimizing error on a test set) does not completely align with what is needed in real-world situations. Therefore, the need arises to better understand how the model works. Some of the main reasons are:

- **Trust:** To ensure that the model will perform well in real-world situations.
- **Causality:** To understand whether a cause truly exists behind the outcomes, not merely a correlation.
- **Transferability:** To be able to apply what has been learned to new cases.
- **Decision support:** So that the model not only provides results but also helps make better decisions.
- **Ethical aspects:** To avoid unfair decisions or discrimination.

A model becomes interpretable when it exhibits transparency, which seeks to make the model easy to understand internally and can be evaluated at three levels:

- **Simulatability:** That a person can understand the entire model without difficulty.
- **Decomposability:** That each part of the model has a clear meaning.
- **Algorithmic transparency:** That it is understood how the model learns or is trained.

Based on these criteria, this research selected those models that meet the interpretability characteristics according to the previous results of the Systematic Literature Review, which are presented below.

Decision Tree

Decision trees are supervised classification algorithms that organize the learning process into a hierarchical structure composed of internal nodes that evaluate attributes, branches that represent conditions, and leaves that indicate the final class. The selection of each attribute is performed based on impurity reduction criteria, such as entropy or the Gini index [52,53].

The construction procedure is recursive: in each iteration, the variable that produces the greatest impurity reduction with respect to the target class is selected, and the process continues until a predefined stopping criterion is reached.

In Software Defect Prediction (SDP), these models have been applied to establish relationships between software metrics and the occurrence of defects [2] developed predictors based on static metrics using

tree-based data mining techniques. Complementarily, Ref. [40] reported that this approach yields results comparable to other classifiers across several defect datasets.

Naïve Bayes

The Naïve Bayes model is a probabilistic classification algorithm that applies Bayes' theorem to estimate the posterior probability of a class A given a set of features B :

$$P(A | B) = \frac{P(A) \times P(B | A)}{P(B)}$$

Its main hypothesis establishes that the predictor variables are conditionally independent of each other once the class is known [54].

In software defect prediction, Naïve Bayes has been frequently adopted due to its computational efficiency and stable behavior in the presence of datasets with numerous potentially correlated metrics. The systematic review by [12] identifies it as one of the most widely used classifiers in SDP studies. Likewise, Ref. [40] include it in their comparative analysis, reporting consistent results across various projects.

Logistic Regression

Logistic regression is a statistical model designed to estimate the probability of occurrence of a binary event from a set of explanatory variables. Unlike linear regression, this model employs the logistic or sigmoid function to transform the linear combination of independent variables into a value between 0 and 1, allowing the result to be directly interpreted as a probability [55].

$$P(Y = 1 | X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \dots + \beta_n X_n)}}$$

In software defect prediction, logistic regression has been used to quantitatively analyze the relationship between software metrics and historical defects [2] describe it as one of the most widely used traditional methods in this field. Similarly, Ref. [40] include it within their benchmarking framework, evidencing its performance across various defect datasets.

4.3.2 Black-Box Models

Black-box algorithms are defined in the literature as models that are difficult to interpret for human beings. This term is used to describe systems where, although inputs and outputs can be observed, the internal mechanism by which the model reaches a decision is unclear or cannot be intuitively grasped [51].

According to the literature, black-box algorithms are primarily characterized by being difficult to understand. That is, it is unclear how they arrive at their results. One of their main characteristics is the lack of transparency. This means that a person cannot fully understand how the model works nor easily explain each of its parts. Furthermore, their internal processes are not clear. Although the model can make predictions, it is not possible to easily see how it made those decisions. This is common in more complex models, such as deep learning [51].

Unlike simpler models, which can be understood step by step, black-box algorithms have very high complexity. For this reason, they surpass a person's ability to analyze them completely. Due to this, post-hoc interpretation techniques are often used, which allow explaining the model's results after it has been trained,

without needing to understand its entire internal functioning. Based on these characteristics, this research selected several algorithms considered as black-box, which are presented below.

Random Forest

According to [56], the Random Forest algorithm is a classifier based on an ensemble of decision trees, where each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

For its construction, multiple individual trees are generated by injecting randomness; that is, for each tree (the k -th), a random vector Θ_k , independent of previous ones, is defined, and the model is trained using this vector. During the growth of each tree, additional variability is introduced by randomly selecting a subset of input variables at each node to determine the best partition, instead of using all available variables. Additionally, this algorithm employs bagging, where each tree is trained with a random bootstrap sample (with replacement) from the original dataset, which helps reduce the model's variance.

Once the ensemble of trees is built, the algorithm operates as a voting system. In classification problems, each tree casts a vote, and the final prediction corresponds to the majority class, whereas in regression problems, the result is obtained as the average of individual predictions. Among the key concepts of its functioning is the convergence of the generalization error, which tends to stabilize as the number of trees increases, thereby reducing the risk of overfitting. Furthermore, the model's performance depends on two main factors: the accuracy of individual trees and the low correlation among them. Another relevant aspect is the use of out-of-bag estimates, which allow evaluating the model's error using the samples not selected during the bagging process, without requiring an additional validation set. Finally, Random Forest enables the estimation of variable importance by analyzing the impact on the error when these variables are randomly altered.

According to [56,57], this algorithm has several advantages that explain why it is widely used. One of its main advantages is that it does not tend to overfit easily; in other words, it does not “memorize” the training data but rather generalizes well to new data. This is because it uses many decision trees and combines their results. Furthermore, it has high accuracy and is robust to noise and outliers, meaning that it can continue to perform well even when data are imperfect. It is also a fast and efficient method, as it can be trained in less time than other similar algorithms and can be executed in parallel.

Another important advantage is that it can handle complex data with many variables, without requiring extensive prior adjustments. It can even identify which variables are most important for prediction, providing useful information to the user.

However, despite these advantages, Random Forest also has some limitations. The most significant is that it is difficult to interpret, since being composed of many trees, it is challenging to understand exactly how it makes a decision. For this reason, it is considered a “black-box” model. Moreover, its performance depends on a balance between how accurate the individual trees are and how different they are from each other; if this balance is not appropriate, accuracy may be affected.

Gradient Boosting

The Gradient Boosting algorithm is a machine learning technique that seeks to improve model accuracy by combining several simple models, typically small decision trees. According to [56], this method builds the model iteratively, where each new model attempts to correct the errors made by the previous ones. Rather than working directly on the data, the algorithm focuses on optimizing a loss function, that is, on reducing prediction error step by step.

The process begins with a simple initial estimate, usually a constant value. From there, at each iteration, the errors of the current model, known as “pseudo-responses”, are calculated, indicating how far the predictions are from the actual values. Then, a new model (weak learner) is trained to predict these errors and is added to the previous model to improve predictions. This process is repeated several times, adjusting each new model with a factor called the learning rate, which helps prevent the model from overfitting.

Among its main advantages, Gradient Boosting stands out for its high accuracy and its ability to compete with other advanced algorithms in both classification and regression tasks. Additionally, it is a flexible method, as it can be adapted to different loss functions depending on the problem. It is also robust to imperfect data, including outliers, and can handle complex datasets with many variables, even automatically identifying which are most important. Another advantage is that it can handle missing data and does not require complex transformations of input variables.

However, this algorithm also has some disadvantages. One of the main drawbacks is that it can be prone to overfitting if too many iterations are used, although this can be controlled by the learning rate. Furthermore, its computational cost is high, as it requires training multiple models sequentially, which can make it slow compared to other methods. Moreover, like other ensemble models, it exhibits low interpretability and is therefore often considered a black-box model. Finally, although very powerful, it may not be the best choice when simple interpretation is required or when computational resources are limited.

To address this interpretability difficulty, additional tools such as variable importance measures and partial dependence plots are often used, which allow for a better understanding of the model's behavior without needing to directly analyze its internal structure.

Support Vector Machine (SVM)

The Support Vector Machine (SVM) algorithm is a supervised machine learning method primarily used for classification tasks, standing out for its effectiveness with both linear and non-linear data. Its functioning is based on finding an optimal decision boundary that best separates different data classes.

In simple terms, SVM seeks a hyperplane, that is, a line (in two dimensions) or a plane (in higher dimensions), that separates classes by maximizing the distance between them. The data points closest to this boundary are known as support vectors, and they are the most important, as they determine the hyperplane's position. When data cannot be separated linearly, the algorithm uses functions called kernels to transform the data into a higher-dimensional space where they can be separated. Furthermore, its performance depends on parameters such as C , which controls the trade-off between accuracy and model simplicity, and γ (gamma), which defines how flexible the decision boundary is.

Among its main advantages, SVM stands out for its high accuracy and robustness. It is also a stable and consistent method, as it maintains similar performance even when the training data changes. Another important advantage is its ability to handle imbalanced data, as well as its good generalization capability, meaning it can yield good results on new data.

Nevertheless, SVM also has some limitations. One of them is the complexity of its configuration, as the appropriate choice of parameters such as C and γ can be difficult and requires experimentation. Additionally, there are constraints in some software tools, where it is not always possible to adjust these parameters, limiting performance. There may also be limitations regarding the size of training samples, which can affect its application to large datasets. Finally, its performance may depend on sampling design, especially in complex problems, where careful selection of training data is required.

Multilayer Perceptron (MLP)

The Multilayer Perceptron (MLP), also known as a feedforward neural network, is one of the most representative models of deep learning. According to [58], this model can be understood as a mathematical function that transforms an input dataset into an output by combining multiple simpler operations.

Its functioning is based on a structure organized into layers: an input layer, one or more hidden layers, and an output layer. Information flows in a single direction, from input to output, in a process known as forward propagation. At each layer, data are transformed through mathematical operations that include weights, biases, and non-linear activation functions, such as ReLU or sigmoid, allowing the model to learn complex patterns. During training, the backpropagation algorithm is used, which adjusts the model's parameters by calculating the error and propagating it backward through the network, thus optimizing performance using methods such as gradient descent.

Among their main advantages, MLPs stand out for their ability to approximate complex functions, meaning they can represent a wide variety of relationships among data. Additionally, they have the capacity to automatically learn relevant features without needing them to be manually defined. They also allow building deep representations of data, which improves their generalization capability and enables them to solve complex problems more efficiently.

However, these models also have important disadvantages. One of the main drawbacks is the difficulty of interpretation, as the internal process by which they generate a prediction is highly complex; therefore, they are often considered “black-box” models. Furthermore, their training involves solving non-convex optimization problems, which can make finding the optimal solution challenging. They also require high computational cost, especially when deep networks are used, which implies the use of specialized hardware such as GPUs.

4.4 Parameters for the Algorithms Used

Table 8 presents in detail the configuration parameters used for each of the seven machine learning algorithms employed in the experiments, including both naturally explainable models and black-box models.

Table 8: Configuration parameters by algorithm.

Naturally Explainable Models	
Algorithm	Parameters
Decision Tree	Criterion: Gini
	Max Depth: none
	Max leaf nodes: none
	Min impurity decrease: 0.0
	Random state: 42
Logistic Regression	Class weight: none
	Preprocessing: StandardScaler
	Max iter: 500
	Solver: lbfgs
	Penalty: l2
Naïve Bayes	C: 1.0
	Variant: GaussianNB

(Continued)

Table 8 (continued)

Naturally Explainable Models	
Algorithm	Parameters
	Hyperparameters: default
Black-box Models	
Algorithm	Parameters
Random Forest	Criterion: Gini N estimators: 200 Max Depth: None Max leaf nodes: None Min impurity decrease: 0.0 Class weight: None Random state: 42
Gradient Boosting	N estimators: 200 Learning rate: 0.1 Criterion: Friedman MSE Min impurity decrease: 0.0 Max depth: 3 Random state: 42
SVM	Kernel: RBF Preprocessing: StandardScaler Probability: true C: 1.0 Gamma: scale Random state: 42
MLP	Preprocessing: StandardScaler Max iter: 500 Hidden layer: (100) Random state: 42

A common aspect to all classifiers involving randomness is the use of a fixed random seed, which ensures that data partitioning and experiment execution are fully reproducible. This guarantees that any observed differences in performance can be attributed to the inherent characteristics of each algorithm and not to variations arising from randomness. The specific value of the seed has no statistical implication; any fixed value would yield equally reproducible results.

Regarding the number of estimators, a value of 200 was established for Random Forest, seeking a balance between model stability and computational cost. This value was considered viable because, according to the Strong Law of Large Numbers, the generalization error converges as the number of trees increases without increasing the risk of overfitting [56]. For models requiring variable standardization, such as SVM and MLP, prior data scaling was applied using `StandardScaler` to ensure that all variables had a similar scale, preventing those with larger values from disproportionately influencing the model. The remaining

hyperparameters were kept at their default values, aiming to evaluate the general behavior of each model without excessively tuning it.

5 Model Evaluation Metrics

This section presents the metrics used for evaluating the models developed in this study. Since the problem addressed corresponds to a supervised classification task in the context of software defect prediction, it is necessary to analyze both the predictive performance of the models and the quality of the explanations generated by Explainable Artificial Intelligence methods. Therefore, two groups of metrics are considered: on the one hand, those aimed at measuring model accuracy, and on the other hand, those intended to evaluate the complexity and interpretability of the explanations. This combination allows obtaining a comprehensive view of model behavior, not only in terms of accuracy but also in their ability to generate reliable and understandable explanations.

5.1 Accuracy Metrics

In supervised classification tasks, evaluating model effectiveness requires comparing the predictions made with the true labels of the data. For this purpose, metrics based on the confusion matrix are used, which summarizes the classifier's hits and errors through four values:

- **TP (True Positives):** cases correctly identified as positive.
- **TN (True Negatives):** cases correctly identified as negative.
- **FP (False Positives):** negative cases incorrectly classified as positive.
- **FN (False Negatives):** positive cases incorrectly classified as negative.

These metrics allow understanding not only how many predictions were correct but also which types of errors predominate, which is important in applications where certain errors may be more costly than others [59,60].

5.1.1 Accuracy

Accuracy measures the percentage of correct predictions made by the model over the total number of instances evaluated, considering both the positive and negative classes. Its formula is as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

This metric is useful for obtaining an overall view of performance when classes are balanced. However, it can be misleading in imbalanced datasets, as a model could achieve high accuracy simply by predicting the majority class most of the time [43].

5.1.2 Precision

Precision indicates what proportion of instances classified as positive are actually positive. Its formula is as follows:

$$\text{Precision} = \frac{TP}{TP + FP}$$

This metric is particularly relevant when false positives are costly or undesirable, such as in systems where an incorrect alert could generate unnecessary expenses or wrong decisions [61].

5.1.3 Recall

Recall, also called sensitivity, measures the model's ability to correctly identify all actual positive instances. Its formula is as follows:

$$\text{Recall} = \frac{TP}{TP + FN}$$

High recall is critical in contexts where false negatives represent a significant risk, for example, in the detection of critical software defects, diseases, or failures in industrial systems [62].

5.1.4 F1-Score

The F1-score combines precision and recall into a single metric using the harmonic mean, offering a balance between both indicators. Its formula is as follows:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

By using the harmonic mean, the F1-score strongly penalizes cases where either metric (precision or recall) is very low. For this reason, it is especially useful in imbalanced datasets, where a model could have high accuracy but fail to detect the minority class [63].

5.2 Proposed Metrics of Interpretability in the Study

One of the main challenges in the context of XAI is its evaluation, since interpretability is not a directly measurable property, nor is there a general consensus on how to define it. According to various studies, interpretability is not a single concept but a multidimensional construct that may encompass elements such as fidelity to the model, stability in response to data changes, consistency of explanations, model simplicity, and its comprehensibility to humans [46,64,65].

To ground this evaluation approach in established theory, the framework proposed by Doshi-Velez and Kim [64] was adopted, who distinguish three levels of explainability assessment: application-grounded evaluation, which involves real human subjects performing actual tasks; human-grounded evaluation, which involves simplified human experiments; and functionally-grounded evaluation, which uses quantitative proxies for interpretability without direct human involvement. The proposed metrics (fidelity, stability, and complexity) fall within the functionally-grounded category, meaning they provide rigorous, reproducible, and theory-aligned quantitative measures that serve as proxies for interpretability in the absence of human-subject studies.

Fidelity is grounded in the principle that a high-quality explanation must accurately reflect the behavior of the original black-box model [7]. This aligns with Doshi-Velez and Kim's notion that functionally grounded metrics should measure whether the explanation captures the underlying model's decision logic. Stability, in turn, addresses the concern that explanations may be sensitive to minor data perturbations or training variations, a property that has been identified as critical for trustworthiness in XAI systems [46,66]. Complexity, defined as the proportion of active features in an explanation, reflects the cognitive principle that simpler explanations are more comprehensible to humans, a concept supported by both psychological research on cognitive load and XAI literature on explanation simplicity [6,51].

While we acknowledge that functionally-grounded metrics do not fully capture human interpretability, they provide a necessary and theoretically-grounded foundation for comparing XAI techniques in the SDP context. This approach enables objective, reproducible evaluation across different models and datasets,

which is essential for establishing baseline comparisons before progressing to more resource-intensive human-grounded or application-grounded validation.

Under this approach, the evaluation of methods such as LIME, SHAP, and Breakdown requires analyzing different dimensions that reflect specific properties of the explanation. In general terms, an explanation can be considered adequate if: it faithfully represents the behavior of the base model, maintains consistency in response to variations in the data or training process, and presents a sufficiently simple structure to facilitate understanding.

Based on these criteria, this work has opted to evaluate using the following metrics: Fidelity, Stability, Sparsity, and a Composite Interpretability Score.

5.2.1 Fidelity

Fidelity represents the degree to which an explanation reproduces the behavior of the original model. In local methods such as LIME or SHAP, it involves evaluating how close the explanatory approximation $g(x)$ is to the complex model $f(x)$ in the neighborhood of a given instance [7,46].

Formally, fidelity it can be understood as a measure of the approximation error between the base model (black-box model) and the explanatory model [66]. In this work, fidelity was measured as one minus the absolute difference between the accuracy of the base model and that of the explanatory model in each cross-validation fold, such that values close to 1 indicate high fidelity.

Fidelity equation is shown in (1), where $\text{Acc}_{\text{model}}(k)$ indicates the accuracy of the original model in fold k , while $\text{Acc}_{\text{XAI}}(k)$ represents the accuracy achieved by the explanatory model (pos-hoc) on the same validation partition.

Values close to 1 indicate high fidelity, suggesting that the explanation adequately preserves the internal logic of the base model. In SDP tasks, fidelity acquires special relevance because explanations generated by post-hoc methods can directly influence decision-making related to the prioritization of modules for testing and maintenance. Several authors in the field of XAI have pointed out that an explanation with low fidelity can generate a false perception of understanding of the model, since it does not adequately reflect the actual mechanism leading to the prediction [6,64].

$$\text{Fidelity}_k = 1 - |\text{Acc}_{\text{model}}(k) - \text{Acc}_{\text{XAI}}(k)| \quad (1)$$

In this work, we adopt a global notion of fidelity. We focus on a global metric because instance-level fidelity can be highly sensitive to local fluctuations and outliers: two models may exhibit high pointwise agreement yet differ substantially in generalization performance. By relying on dataset-level accuracy, Eq. (1) provides a more robust assessment of the alignment between the models' behaviors. It should therefore be interpreted as a measure of performance-level (global) alignment, complementary to, but not replacing, instance-level fidelity notions commonly used in XAI.

5.2.2 Stability

Stability evaluates whether explanations remain consistent in response to small variations in the data or training. Ideally, an explanatory method should produce similar results when the base model changes slightly due to different training partitions [66]. However, studies have shown that some post-hoc methods are sensitive to these variations [46].

In this work, stability was measured based on the variance of the accuracy values obtained in cross-validation and its estimation is calculated as shown in (2).

$$\text{Stability} = 1 - \text{Var}(\text{Accuracy}_k) \quad (2)$$

The variance is calculated over the accuracy values obtained in the different cross-validation folds. Statistically, variance is a measure of dispersion that captures the spread of an estimator around its mean across different partitions of the dataset. Low variance values suggest more consistent behavior of the explanatory method, which can be interpreted as a greater level of stability.

Stability has been recognized in the XAI literature as a desirable property, since explanations sensitive to small perturbations may capture noise rather than the actual behavior of the model [6,46,66].

In the context of SDP, this characteristic is considered relevant because unstable explanations might identify different metrics as “critical” in each model run, leading to reduced reliability of the obtained results.

5.2.3 Complexity

Within the framework of interpretability, complexity refers to the proportion of active features within an explanation. According to [6], explanations that rely on a reduced number of variables tend to be more comprehensible, as they limit the amount of information the user must process.

Complexity is defined as (3), where lower values indicate greater structural simplicity, as the explanation depends on a reduced subset of variables.

$$\text{Complexity} = \frac{\# \text{Relevant Features}}{\# \text{Total Features}} \quad (3)$$

In the context of defect prediction, characterized by the presence of various software metrics, complexity proves useful since it allows highlighting a reduced set of key factors on which to focus improvement actions.

However, complexity as a component of the composite interpretability score was not included. This decision requires clarification. For the additive post-hoc explanatory methods implemented in this work (specifically LIME [7] and SHAP [44]) the explanatory structure does not change qualitatively between runs. The explanation is limited to a set of individual additive contributions associated with each variable, without branching rules, hierarchical depth, or symbolic components whose organization varies between instances. The structure remains fixed; only the number of variables with active or relevant contributions may vary.

5.2.4 Composite Interpretability Score

Interpretability does not manifest as a simple, direct attribute; rather, it emerges from the combination of different criteria that allow evaluating how comprehensible a model is, and its evaluation cannot be reduced to a single metric. Various authors have pointed out that explainability involves multiple complementary properties, such as fidelity, stability, simplicity, and comprehensibility [6,46,64].

In [64] it is argued that explanatory systems should not be evaluated with a single metric, but from a multi-criteria perspective that considers different dimensions of the explanatory phenomenon. Therefore, relying on a single metric is insufficient to represent their global quality.

From this perspective, and considering the decision to exclude complexity from the composite metric (as justified in the previous subsection), this work defines the interpretability score as the average of fidelity and stability, as shown in Eq. (4):

$$\text{Interpretability} = \frac{\text{Fidelity} + \text{Stability}}{2} \quad (4)$$

The decision to average both components, rather than summing them directly, responds to the need to maintain the composite metric within the same normalized range $[0, 1]$ as its individual components. Fidelity and stability are considered complementary: fidelity measures correspondence with the model, while stability measures robustness against variations. Their average provides a balanced, single-value indicator of explanation quality that addresses two independent dimensions of interpretability.

6 Results

6.1 Model Accuracy Evaluation

The results obtained in this experimentation regarding the accuracy metrics implemented to evaluate ML algorithms, both naturally explainable and black-box models, are presented below.

The metrics of accuracy, precision, recall, and F1-score are analyzed for each of the five datasets, comparing the performance of naturally explainable models, Decision Tree (DT), Logistic Regression (LR), and Naïve Bayes (NB), against black-box models, Random Forest (RF), Gradient Boosting (GB), Support Vector Machine (SVM), and Multilayer Perceptron (MLP).

For the CM1 dataset, the results are presented in [Table 9](#). Analyzing the F1-score, which reflects a balance between precision and detection capability, the best performance is achieved by Logistic Regression and SVM, both tied at 0.546. This tie is particularly noteworthy, as it indicates that a naturally explainable model (Logistic Regression) matches the performance of a black-box model (SVM) in terms of F1-score. The next best F1-score corresponds to MLP (0.519), followed by Random Forest (0.375), Naïve Bayes (0.323), Decision Tree (0.258), and Gradient Boosting (0.250).

Table 9: Results for the CM1 dataset by metric.

Accuracy	Precision	Recall	F1
MLP (0.902)	MLP (0.539)	Logistic Regression (0.857)	Logistic Regression (0.546)
Logistic Regression (0.850)	Logistic Regression (0.400)	SVM (0.857)	SVM (0.546)
SVM (0.850)	SVM (0.400)	MLP (0.500)	MLP (0.519)
Random Forest (0.850)	Random Forest (0.333)	Random Forest (0.429)	Random Forest (0.375)
Naïve Bayes (0.842)	Naïve Bayes (0.294)	Naïve Bayes (0.357)	Naïve Bayes (0.323)
Decision Tree (0.827)	Decision Tree (0.235)	Decision Tree (0.286)	Decision Tree (0.258)
Gradient Boosting (0.820)	Gradient Boosting (0.222)	Gradient Boosting (0.286)	Gradient Boosting (0.250)

Regarding accuracy and precision, MLP achieves the highest values (0.902 and 0.539, respectively), although this does not translate into a superior F1-score due to its lower recall compared to Logistic Regression and SVM. These results suggest that, for the CM1 dataset, the performance advantage of black-box models is not evident, and explainable models remain competitive, particularly when the harmonic mean of precision and recall is the primary evaluation criterion.

Regarding recall, which indicates the proportion of defective modules correctly identified, Logistic Regression and SVM present the highest values in the dataset (0.857), showing a better capacity for defect detection. On the other hand, MLP achieves a recall of 0.50, indicating a prioritization of precision over detection, identifying fewer defective modules in exchange for reducing false positives.

The Gradient Boosting (0.250) and Decision Tree (0.258) models obtained the lowest F1-scores in the dataset, evidencing that their performance was limited in identifying defects in this dataset.

The results for the JM1 dataset are presented in Table 10. This dataset shows the lowest performance among all the datasets analyzed, with F1-scores not exceeding 0.380 for any of the models. Among the black-box models, SVM achieves the best F1-score (0.380), followed by Gradient Boosting (0.360). In the explainable models group, Logistic Regression obtains the best F1 (0.350), differing by only 0.03 from the best black-box model, indicating that a naturally explainable model approaches the performance of black-box models.

Table 10: Results for the JM1 dataset by metric.

Accuracy	Precision	Recall	F1
Random Forest (0.801)	MLP (0.539)	Naïve Bayes (0.840)	SVM (0.380)
Gradient Boosting (0.759)	Logistic Regression (0.400)	Logistic Regression (0.824)	Gradient Boosting (0.360)
MLP (0.747)	SVM (0.400)	SVM (0.824)	Logistic Regression (0.350)
SVM (0.747)	Random Forest (0.400)	Decision Tree (0.706)	MLP (0.340)
Logistic Regression (0.740)	Decision Tree (0.333)	MLP (0.647)	Random Forest (0.310)
Decision Tree (0.727)	Gradient Boosting (0.308)	Random Forest (0.647)	Decision Tree (0.310)
Naïve Bayes (0.299)	Naïve Bayes (0.175)	Gradient Boosting (0.647)	Naïve Bayes (0.300)

In JM1, Naïve Bayes exhibits the highest recall in the dataset (0.840), at the cost of a very low accuracy (0.299) and a reduced F1 (0.300). This indicates a tendency to classify a large proportion of modules as defective, increasing recall but deteriorating the other metrics. Meanwhile, Random Forest achieves the best accuracy in the dataset (0.801) and the best precision among black-box models (0.400), although its recall (0.647) limits its F1 to 0.310.

As shown in Table 11, in KC1, the model with the best overall F1-score is Logistic Regression (0.3677), being the only one exceeding a value of 0.35. This demonstrates that a naturally explainable model has outperformed black-box models. The black-box models with the best results are SVM (0.3396) and MLP (0.3377), followed by Random Forest (F1 = 0.3304).

Table 11: Results for the KC1 dataset by metric.

Accuracy	Precision	Recall	F1
Random Forest (0.799)	Random Forest (0.380)	Naïve Bayes (0.862)	Logistic Regression (0.368)
Gradient Boosting (0.794)	Gradient Boosting (0.365)	Logistic Regression (0.631)	Decision Tree (0.350)
Decision Tree (0.757)	Decision Tree (0.321)	SVM (0.554)	SVM (0.340)
MLP (0.692)	MLP (0.266)	MLP (0.462)	MLP (0.338)
SVM (0.635)	Logistic Regression (0.260)	Decision Tree (0.385)	Random Forest (0.330)
Logistic Regression (0.632)	SVM (0.245)	Random Forest (0.292)	Gradient Boosting (0.325)
Naïve Bayes (0.326)	Naïve Bayes (0.184)	Gradient Boosting (0.292)	Naïve Bayes (0.303)

Naïve Bayes in KC1 presents a pattern similar to JM1: a very high recall (0.8615) with an accuracy of only 0.33, indicating a tendency to overclassify modules as defective in imbalanced datasets. On the other hand, Random Forest and Gradient Boosting achieve the highest accuracies in the dataset but low recall, which limits defect detection.

Regarding the results of KC2 dataset shown in Table 12, the highest F1-score is obtained by Logistic Regression (0.7077), surpassing black-box models. Among the black-box models, the best is Random Forest (F1 = 0.6774), followed by MLP (0.63), SVM (0.62), and Gradient Boosting (0.61). Naïve Bayes achieves the highest precision in the dataset (0.71). Thus far, this dataset shows the most balanced results.

Table 12: Results for the KC2 dataset by metric.

Accuracy	Precision	Recall	F1
Logistic Regression (0.838)	Naïve Bayes (0.714)	Logistic Regression (0.767)	Logistic Regression (0.708)
Random Forest (0.829)	Logistic Regression (0.657)	Random Forest (0.700)	Random Forest (0.677)
Naïve Bayes (0.821)	Random Forest (0.656)	MLP (0.667)	MLP (0.635)
MLP (0.803)	Gradient Boosting (0.621)	SVM (0.667)	SVM (0.625)
Gradient Boosting (0.803)	MLP (0.606)	Gradient Boosting (0.600)	Gradient Boosting (0.610)
SVM (0.795)	SVM (0.588)	Naïve Bayes (0.500)	Naïve Bayes (0.588)
Decision Tree (0.769)	Decision Tree (0.560)	Decision Tree (0.467)	Decision Tree (0.509)

For PC1, corresponding to [Table 13](#), Logistic Regression achieves the best overall F1-score (0.395), followed by Random Forest (0.375), Gradient Boosting (0.343), and MLP (0.324). Among explainable models, Logistic Regression achieves the highest F1-score (0.395), followed by Decision Tree (0.308). Naïve Bayes obtains the lowest F1 (0.222), although it presents a seemingly high accuracy (0.828). PC1 records the highest accuracy, with Random Forest reaching 0.930 and Gradient Boosting 0.919.

Table 13: Results for the PC1 dataset by metric.

Accuracy	Precision	Recall	F1
Random Forest (0.930)	Random Forest (0.400)	Logistic Regression (0.706)	Logistic Regression (0.395)
Gradient Boosting (0.919)	Gradient Boosting (0.333)	SVM (0.529)	Random Forest (0.375)
MLP (0.912)	MLP (0.300)	Naïve Bayes (0.412)	Gradient Boosting (0.343)
Decision Tree (0.905)	Decision Tree (0.274)	Random Forest (0.353)	MLP (0.324)
Naïve Bayes (0.828)	Logistic Regression (0.274)	Gradient Boosting (0.353)	Decision Tree (0.308)
Logistic Regression (0.811)	SVM (0.167)	MLP (0.353)	SVM (0.254)
SVM (0.814)	Naïve Bayes (0.152)	Decision Tree (0.353)	Naïve Bayes (0.222)

Overall, black-box models outperform naturally explainable models in three of the five datasets analyzed (CM1, JM1, and PC1). In the KC1 and KC2 datasets, the explainable model Logistic Regression achieves the best F1-score, surpassing all black-box models.

Regarding per-model performance, MLP stands out as the best black-box model in CM1. Meanwhile, Random Forest achieves the best results among black-box models in KC2 and PC1.

Concerning explainable models, Logistic Regression emerges as the most consistent, according to the results obtained in KC1 and KC2, where it achieved the highest F1-score, surpassing all black-box models.

Given the experimental results obtained across the five datasets, working with black-box models is worthwhile, particularly when the primary objective is to maximize predictive performance. Black-box models outperformed naturally explainable models in three out of five datasets (CM1, JM1, and PC1), demonstrating their superior capacity to capture complex patterns in software defect data. However, in the remaining datasets (KC1 and KC2), the explainable model Logistic Regression achieved the best F1-score, surpassing all black-box models. These results indicate that while black-box models generally offer superior predictive power in most datasets, explainable models remain highly competitive and can even be the best choice in certain contexts. The key challenge associated with black-box models is their lack of interpretability, which can be effectively addressed through post-hoc Explainable AI (XAI) techniques. Therefore, given that black-box models offer equal or superior predictive power in the majority of cases and that their interpretability limitations can be mitigated, it is worthwhile to work with black-box models in this

context, provided that appropriate XAI methods are subsequently applied to ensure transparency and trust in their predictions.

6.2 Model Interpretability Evaluation

Figs. 2–4 graphically present the most influential variables for predicting software defects (both for classifying a module as defective or non-defective). These figures show the results of applying the post-hoc methods LIME, SHAP, and Breakdown to the Random Forest black-box model, using a single fold of the CM1 dataset. Complete results for all black-box models and datasets are available in [67–69].

In Fig. 2, LIME shows the baseline defect probability is 28%. The main negative contributions (pushing the prediction toward “Non-Defective”) are: *locCodeAndComment* (−0.033), *uniq_{Opnd}* (−0.029), and *loc* (−0.024). Conversely, the variable *iv(g)* (design complexity) shows a positive contribution of +0.030, making it the strongest driver toward a “Defective” classification.

In Fig. 3, SHAP is applied to the same CM1 fold, this method identifies *iv(g)* as the variable with the highest positive contribution (+0.06). This indicates that the analyzed module has a high number of independent decision paths in its control flow. Unlike *v(g)*, which measures the overall complexity of the flow graph, *ev(g)* focuses on paths that cannot be structurally simplified, revealing more complex conditional logic. Other variables show positive contributions toward “Defective”: *uniq_{Op}* (+0.02), *b* (+0.02), and *branchCount* (+0.02). Conversely, *total_{Opnd}* (−0.03), *e* (−0.02), and *l* (−0.02) contribute toward “Non-Defective”. Despite these negative contributions, the module is classified as defective because the positive weight of *iv(g)* outweighs the opposing influences.

In the results of Fig. 4, Breakdown is applied to the same fold, this method shows that Random Forest starts from a baseline value of 0.159. This means that, during training, approximately 15.9% of the modules in the CM1 dataset were classified as defective. This value represents the starting point before considering the specific features of the module. From there, variables progressively adjust the prediction until reaching the final value of 0.28.

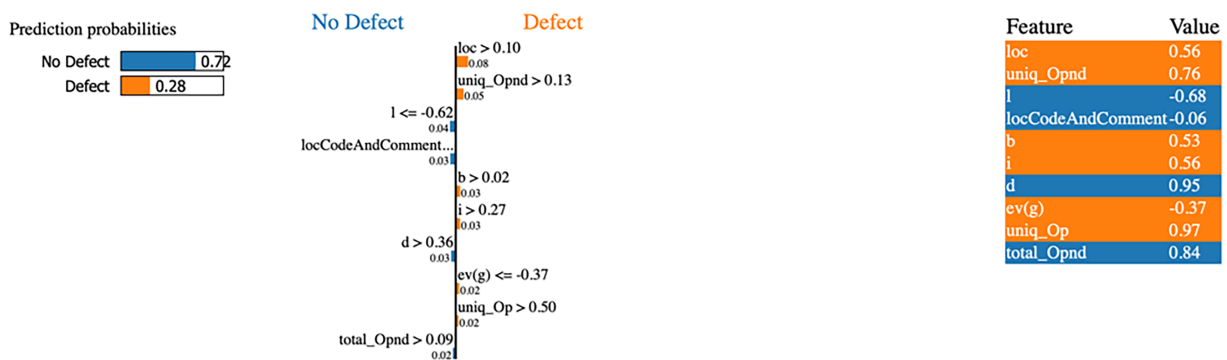


Figure 2: Significant variables obtained by the LIME method for the CM1 dataset and the Random Forest model.

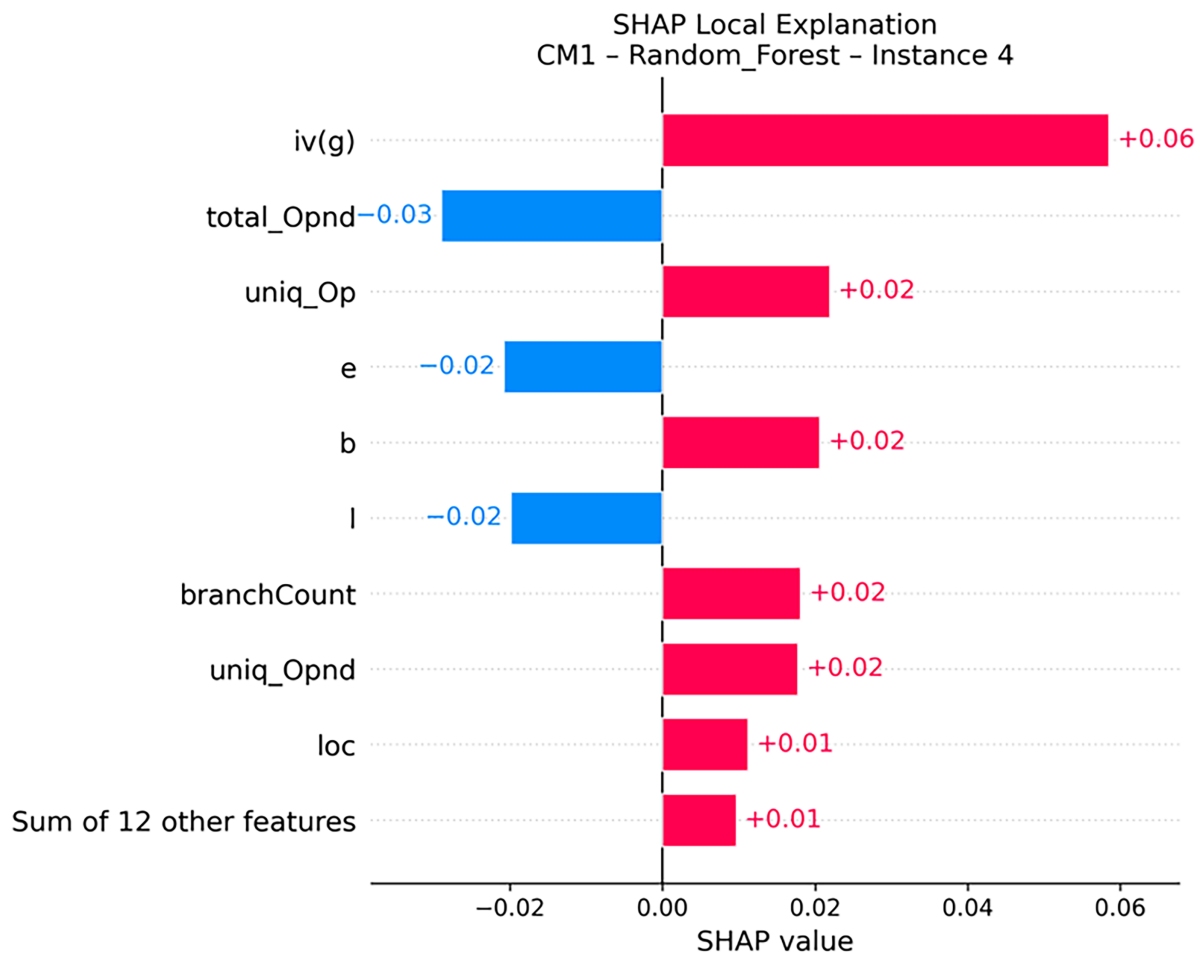


Figure 3: Significant variables obtained by the SHAP method for the CMI dataset and the Random Forest model.

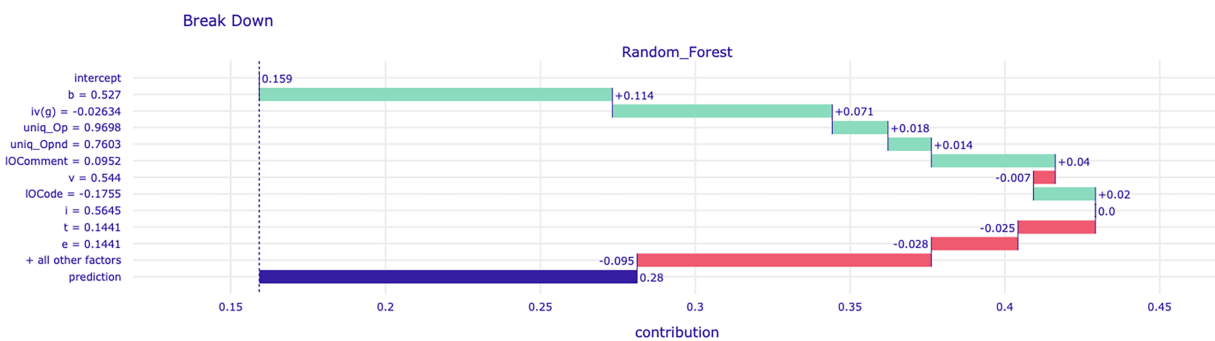


Figure 4: Significant variables obtained by the Breakdown method for the CMI dataset and the Random Forest model.

The most influential variable in this instance is b (number of alternative paths according to Halstead metrics), with a value of 0.527 and a contribution of +0.114. It is important to note that the magnitude of the contribution does not depend directly on the variable's value, but on what the model learned during training within that specific range. Second, $iv(g)$ appears; although its value in this instance is negative (-0.02634), its contribution is positive (+0.071). This occurs because the model does not interpret variables in isolation

but rather based on learned patterns. Next is *uniqOp* (unique operators), with a value of 0.9698 and a modest contribution of +0.018, confirming that a high value does not necessarily imply a high predictive impact. Similarly, *uniqOpnd* (0.7603) contributes +0.014. Finally, *LOComment* (0.0952) and *LOCode* (−0.1755) contribute +0.040 and +0.020, respectively, adding a positive push toward the defective prediction.

This subsection presents the results obtained after applying the explainability metrics defined in Section 5.2. The analysis was performed on black-box models by applying the selected explainability techniques across the five datasets.

6.2.1 Fidelity

Fidelity was the metric with the best results in the experiment. Random Forest, Gradient Boosting, and MLP achieved the maximum value of 1.0 across all datasets and with all three explanation methods, indicating that these techniques successfully replicated the behavior of these models in every instance evaluated. This implies that the explanations produced by LIME, SHAP, and Breakdown can be considered faithful representations of the decision-making process of these models when classifying modules as defective or non-defective.

In contrast, SVM exhibited differentiated behavior, achieving a fidelity of 0.8 across all datasets and explanation techniques. This means that 20% of the instances evaluated with these explanation techniques failed to adequately approximate SVM's behavior; therefore, the explanations generated do not accurately represent the reasons behind its decisions in those instances. Notably, this value was identical across the three techniques and all datasets, suggesting that the model's intrinsic characteristics—particularly the complex non-linear decision boundaries induced by the RBF kernel—may be influencing this outcome, as the linear surrogate models employed by LIME struggle to approximate SVM's behavior faithfully.

Table 14 shows the values obtained when calculating the metric, grouped by explanation technique, applied ML model, and dataset used

Table 14: Fidelity metric results.

Dataset	Model	LIME	SHAP	Breakdown
CM1	Random Forest	1.0	1.0	1.0
	Gradient Boosting	1.0	1.0	1.0
	SVM	0.8	0.8	0.8
	MLP	1.00	1.00	1.00
JM1	Random Forest	1.0	1.0	1.0
	Gradient Boosting	1.0	1.0	1.0
	SVM	0.8	0.8	0.8
	MLP	1.0	1.0	1.0
KC1	Random Forest	1.0	1.0	1.0
	Gradient Boosting	1.0	1.0	1.0
	SVM	0.8	0.8	0.8
	MLP	1.0	1.0	1.0

(Continued)

Table 14 (continued)

Dataset	Model	LIME	SHAP	Breakdown
KC2	Random Forest	1.0	1.0	1.0
	Gradient Boosting	1.0	1.0	1.0
	SVM	0.8	0.8	0.8
	MLP	1.0	1.0	1.0
PC1	Random Forest	1.0	1.0	1.0
	Gradient Boosting	1.0	1.0	1.0
	SVM	0.8	0.8	0.8
	MLP	1.0	1.0	1.0

6.2.2 Stability

Stability evaluates whether explanations remain consistent across different analyzed instances. A value of 1.0 indicates that the explanation method generates completely uniform explanations, meaning that the described behavior is identical regardless of the instance or fold evaluated. A lower value indicates variability among instances, which does not necessarily imply that the explanations are incorrect, but rather that the method is capturing local differences in model behavior. In this context, a low stability result implies that the features identified as relevant may change between instances, reducing the reliability of explanations as a comparative analysis tool.

According to the results, stability was among the metrics with the greatest variability between models, while it remained practically constant across the three explanation methods for each model-dataset combination. This may indicate that the metric is closely related to the base model, which completely determines its results.

Consistent results were observed for some models. For example, Random Forest and Gradient Boosting achieved stability values of 0.84 in CM1, JM1, and KC1, and 0.76 in KC2 and PC1, with identical values across all three explanation techniques.

On the other hand, MLP maintained a constant value of 0.76 across all explanation techniques and all datasets, indicating the lowest stability level among the evaluated models.

Finally, SVM showed the most varied behavior: it achieved a stability of 1.00 in JM1 and KC1, indicating perfectly uniform explanations, but dropped to 0.76 in CM1, KC2 and PC1. This variability suggests that the consistency of SVM's explanations depends on the dataset, which may be related to the complexity of its decision boundaries and the ability of the explanation methods to capture them uniformly across different instances.

[Table 15](#) shows the values obtained when calculating the metric, grouped by explanation technique, applied ML model, and dataset used.

Table 15: Stability metric results.

Dataset	Model	LIME	SHAP	Breakdown
CM1	Random Forest	0.84	0.84	0.84
	Gradient Boosting	0.84	0.84	0.84
	SVM	0.76	0.76	0.76
	MLP	0.76	0.76	0.76
JM1	Random Forest	0.84	0.84	0.84
	Gradient Boosting	0.84	0.84	0.84
	SVM	1.00	1.00	1.00
	MLP	0.76	0.76	0.76
KC1	Random Forest	0.84	0.84	0.84
	Gradient Boosting	0.84	0.84	0.84
	SVM	1.00	1.00	1.00
	MLP	0.76	0.76	0.76
KC2	Random Forest	0.76	0.76	0.76
	Gradient Boosting	0.76	0.76	0.76
	SVM	0.76	0.76	0.76
	MLP	0.76	0.76	0.76
PC1	Random Forest	0.76	0.76	0.76
	Gradient Boosting	0.76	0.76	0.76
	SVM	0.76	0.76	0.76
	MLP	0.76	0.76	0.76

6.2.3 Complexity

Complexity indicates how many variables are used within an explanation compared to the total available. A value of 1.0 means that all variables are used, while lower values indicate that only a subset is employed. In defect prediction, lower complexity is preferable because it allows for easy identification of which metrics are truly important without having to analyze too much information simultaneously.

In the results, complexity was the only metric that distinguished between explanation methods, as it remained the same regardless of the model or dataset. LIME obtained a value of 0.476 in all cases, meaning it uses approximately half of the variables and focuses only on the most relevant ones. In contrast, SHAP and Breakdown had a value of 1.0, indicating that they consider all variables in their explanations.

[Table 16](#) shows the values obtained when calculating the metric, grouped by explanation technique, applied ML model, and dataset used.

6.2.4 Composite Interpretability Score

The composite interpretability score combines fidelity and stability into a single value between 0 and 1. A value close to 1 means that the explanations are reliable because they accurately represent the model's behavior and are consistent across different cases. In contrast, a low value indicates that the explanations fail in one of these aspects, reducing their usefulness for decision-making, such as identifying defective modules.

Table 16: Complexity metric results.

Dataset	Model	LIME	SHAP	Breakdown
CM1	Random Forest	0.476	1.00	1.00
	Gradient Boosting	0.476	1.00	1.00
	SVM	0.476	1.00	1.00
	MLP	0.476	1.00	1.00
JM1	Random Forest	0.476	1.00	1.00
	Gradient Boosting	0.476	1.00	1.00
	SVM	0.476	1.00	1.00
	MLP	0.476	1.00	1.00
KC1	Random Forest	0.476	1.00	1.00
	Gradient Boosting	0.476	1.00	1.00
	SVM	0.476	1.00	1.00
	MLP	0.476	1.00	1.00
KC2	Random Forest	0.476	1.00	1.00
	Gradient Boosting	0.476	1.00	1.00
	SVM	0.476	1.00	1.00
	MLP	0.476	1.00	1.00
PC1	Random Forest	0.476	1.00	1.00
	Gradient Boosting	0.476	1.00	1.00
	SVM	0.476	1.00	1.00
	MLP	0.476	1.00	1.00

In the results, Random Forest and Gradient Boosting achieved the highest and most consistent composite scores. Both models presented similar values across all evaluated combinations. In CM1, JM1, and KC1, they achieved a score of 0.92, resulting from combining a fidelity of 1.0 with a stability of 0.84. This indicates that the explanations generated for these models in these datasets are faithful to the model's behavior and consistent across instances, becoming reliable tools for identifying the software metrics that influenced module classification.

For KC2 and PC1, the score dropped to 0.88 as a consequence of lower stability in these datasets. Although the difference is small, it reflects significant variability among instances. Since fidelity remains at 1.0, the explanations are still correct at the individual level, thus remaining useful for specific instance analysis.

MLP obtained a composite score of 0.88 across all datasets and was the only model whose value did not change. This may indicate that MLP's explanations maintain good consistency, although they may not be the most optimal. Nevertheless, it reflects stability in its results regardless of the dataset or explanation technique implemented.

SVM, on the other hand, presented a wider range of scores, showing three distinct behaviors. In JM1 and KC1, it achieved the maximum value of 1.00, indicating completely faithful and fully consistent explanations across instances. This represents one of the best results obtained, making SVM, for these datasets, an explainable model with superior results. In KC2, SVM obtained a score of 0.88, equivalent to that of MLP and the other models in this dataset. In this case, the result is due to a combination of fidelity valued at 1.0 and stability valued at 0.76, indicating correct explanations but with some variability among instances.

Finally, in CM1 and PC1, SVM achieved the lowest value of the experiment at 0.78, resulting from a fidelity of 0.8 and a low stability of 0.76. This means that the explanations not only partially fail to represent the model's actual behavior but also vary among instances, preventing the identification of consistent patterns regarding the most relevant metrics. Therefore, these specific datasets do not inspire complete confidence in the explanation results.

Table 17 shows the values obtained when calculating the metric, grouped by explanation technique, applied ML model, and dataset used.

Table 17: Interpretability metric results.

Dataset	Model	LIME	SHAP	Breakdown
CM1	Random Forest	0.92	0.92	0.92
	Gradient Boosting	0.92	0.92	0.92
	SVM	0.78	0.78	0.78
	MLP	0.88	0.88	0.88
JM1	Random Forest	0.92	0.92	0.92
	Gradient Boosting	0.92	0.92	0.92
	SVM	0.90	0.90	0.90
	MLP	0.88	0.88	0.88
KC1	Random Forest	0.92	0.92	0.92
	Gradient Boosting	0.92	0.92	0.92
	SVM	0.90	0.90	0.90
	MLP	0.88	0.88	0.88
KC2	Random Forest	0.88	0.88	0.88
	Gradient Boosting	0.88	0.88	0.88
	SVM	0.88	0.88	0.88
	MLP	0.88	0.88	0.88
PC1	Random Forest	0.88	0.88	0.88
	Gradient Boosting	0.88	0.88	0.88
	SVM	0.78	0.78	0.78
	MLP	0.88	0.88	0.88

6.3 Statistical Analysis

To statistically validate the performance comparisons, we applied the following procedure to all evaluated models, both in terms of predictive accuracy and model complexity. For each dataset and metric, we obtained $k = 5$ values from a five-fold cross-validation.

First, we assessed whether these five-fold results followed a normal distribution at a 95% confidence level. To this end, we applied two normality tests: the Shapiro–Wilk test and kurtosis test. These tests were used to determine whether the null hypothesis of normality could be accepted.

Since normality was not always satisfied, we first checked the results of the three normality tests. When all three tests did not reject the normality assumption, we applied a parametric one-way ANOVA (at a 95% confidence level) to compare the means across models. If at least one of the normality tests indicated a violation of normality, we instead employed the non-parametric Friedman test (also at a 95% confidence

level) to detect whether there was at least one statistically significant difference in medians among the compared models across the k-fold results.

The detailed results of these statistical tests for the accuracy-based metrics, broken down by dataset, can be found in [70], while the corresponding results for the interpretability-related metrics are reported in [71]. As a remark, no statistical tests were applied to the complexity metric because the results were identical across models, as shown in Table 16.

For those cases where the ANOVA or Friedman test indicated a significant difference, we then performed a post-hoc pairwise analysis between the best and the second-best model by metric.

Specifically, we again tested for normality (using the same three normality tests) on the distribution of the differences between the best and second-best model over the five folds. If the differences were normally distributed, we applied the paired Student's *t*-test as a parametric test; otherwise, we used the Wilcoxon signed-rank test as a non-parametric alternative. In both cases, we tested at the 95% confidence level to determine whether the performance of the best model was statistically superior to that of the second-best.

The results of the post-hoc tests can be found in [72], which report the pairwise comparison between the best and second-best models (as identified in Tables 9–13) for the accuracy-based metrics.

Finally, for Table 17, post-hoc tests were not performed because SHAP, LIME, and BreakDown yielded identical results for each predictive model (i.e., for a given model such as Random Forest, the outcomes did not differ across the post-hoc explanation methods).

6.4 Computational Overhead Analysis

To assess the practical feasibility of our XAI-enhanced SDP framework, we measured the runtime overhead and memory consumption associated with each model and XAI technique. All experiments were executed on a machine with an Dual-Core Intel Core i5 to 2.30 GHz, 8 GB RAM, and macOS Ventura version 13.7.8, using Python 3.8. The measurements include: model training time, explanation generation time for SHAP, LIME, and Breakdown, and peak memory usage during explanation generation. All times are reported in seconds, averaged across the five cross-validation folds.

Table 18 presents the computational costs for each model and XAI technique across all datasets. The symbol “–” indicates that the model is naturally explainable and does not require post-hoc explanation techniques. All times are reported in seconds, averaged across five cross-validation folds. Peak memory refers to the maximum memory usage during explanation generation.

Our analysis reveals three key findings. First, SHAP is significantly more computationally expensive than LIME and Breakdown for all models and datasets. SHAP explanation times ranged from less than 0.01 s for tree-based models to 498 s for SVM on JM1. In contrast, LIME remained consistently under 15 s across all datasets. Second, computational cost scales with dataset size and model complexity. SVM exhibited the highest cost for both training and explanation, consistent with its $O(n^3)$ complexity. Third, memory consumption varies by model and technique. MLP required the most memory (232 MB), while Random Forest and Gradient Boosting required less (28–50 MB) except for PC1 data set. The naturally explainable models (Decision Tree, Logistic Regression, Naïve Bayes) consumed negligible additional memory.

Table 18: Computational overhead analysis: training time, explanation time, and peak memory usage.

Dataset	Model	Training Time (s)	SHAP (s)	LIME (s)	Breakdown (s)	Peak Memory (MB)
CM1	Decision Tree	0.0785	–	–	–	–
	Logistic Regression	0.0989	–	–	–	–
	Naïve Bayes	0.0041	–	–	–	–
	Random Forest	0.9831	0.0065	0.4667	10.1697	29.01
	Gradient Boosting	1.8333	0.0009	0.2461	0.8160	33.24
	SVM	0.1936	24.0353	1.0605	2.3951	74.57
	MLP	5.1319	0.9100	0.2213	1.2709	232.40
JM1	Decision Tree	0.7044	–	–	–	–
	Logistic Regression	0.3410	–	–	–	–
	Naïve Bayes	0.0328	–	–	–	–
	Random Forest	6.3609	0.4229	0.5361	12.7614	28.86
	Gradient Boosting	23.0006	0.0018	0.2223	2.3022	31.59
	SVM	90.8173	498.1472	14.8878	449.7559	74.65
	MLP	39.6967	1.2165	0.2874	1.8017	232.49
KC1	Decision Tree	0.0699	–	–	–	–
	Logistic Regression	0.0656	–	–	–	–
	Naïve Bayes	0.0074	–	–	–	–
	Random Forest	1.6164	0.0862	0.4071	6.0580	28.96
	Gradient Boosting	4.1157	0.0048	0.3119	0.6304	50.40
	SVM	2.0841	65.5123	4.8555	9.0354	74.63
	MLP	7.3321	1.7532	0.2106	0.4246	232.48
KC2	Decision Tree	0.0150	–	–	–	–
	Logistic Regression	0.0288	–	–	–	–
	Naïve Bayes	0.0045	–	–	–	–
	Random Forest	0.8518	0.0069	0.4653	8.6852	28.94
	Gradient Boosting	1.8580	0.0010	0.3798	0.6817	49.99
	SVM	0.0673	14.0671	0.5798	1.2982	74.62
	MLP	2.1875	1.9332	0.2713	0.8384	232.51
PC1	Decision Tree	0.0702	–	–	–	–
	Logistic Regression	0.0369	–	–	–	–
	Naïve Bayes	0.0117	–	–	–	–
	Random Forest	1.3708	0.0192	0.4162	10.2676	87.41
	Gradient Boosting	9.0372	0.0005	0.2005	0.8286	44.15
	SVM	1.0313	35.7147	1.5267	12.8437	74.63
	MLP	6.9268	1.0631	0.1623	1.1039	232.48

These results have important practical implications. In resource-constrained or real-time environments, LIME is preferable over SHAP due to its lower computational cost. For tree-based models, TreeSHAP offers a computationally efficient alternative that retains the theoretical advantages of SHAP. The naturally

explainable models, while slightly less accurate in some datasets, offer the lowest computational overhead, making them suitable for rapid prototyping or resource-constrained deployment scenarios.

7 Discussions

7.1 Comparison with Previous Studies

Our findings align with and extend previous work in software defect prediction and Explainable AI. The observation that naturally explainable models, particularly Logistic Regression, can achieve competitive or superior performance in certain datasets is consistent with the findings of [40], who reported that simpler classifiers can perform as well as more complex ones in software defect prediction contexts. This study extends this finding by demonstrating that Logistic Regression not only achieves competitive accuracy but also provides naturally interpretable explanations without requiring post-hoc techniques, offering a practical advantage in scenarios where interpretability is a priority.

The fidelity and stability results for Random Forest and Gradient Boosting are consistent with previous studies showing that tree-based models are particularly suitable for explanation methods such as SHAP and LIME [7,44]. The near-perfect fidelity we observed for these models aligns with the findings of [6], who noted that ensemble methods tend to produce more stable and faithfully approximable explanations due to their structured decision boundaries. The lower fidelity observed for SVM in this study also aligns with prior work suggesting that kernel-based methods can be more difficult to approximate with local linear models due to their non-linear decision boundaries [46].

Our finding that LIME produces more concise explanations than SHAP and Breakdown is consistent with the design goals of LIME, which explicitly optimizes for simplicity through L1 regularization [7]. However, this study provides novel empirical evidence of this characteristic specifically in the SDP context, quantifying the difference in feature selection across the three XAI techniques. This is particularly relevant for practitioners who must balance explanation completeness against cognitive load for end-users.

Compared to recent work on metaheuristic optimization for SDP [36,37], our study adopts a complementary approach. While those studies focus on optimizing predictive performance through algorithm tuning and feature selection, this work emphasizes interpretability alongside accuracy, providing practitioners with guidance on selecting models that balance both objectives. We believe that future work could productively combine both lines of research by applying metaheuristic optimization to the XAI-SDP framework to further enhance both predictive performance and explanation quality.

7.2 Predictive Performance of Black-Box Models vs. Naturally Explainable Models

The results obtained after applying the accuracy metrics showed that black-box models do not consistently outperform naturally explainable models across all datasets or in all analyzed cases.

In the CM1 dataset, Logistic Regression achieved performance practically equivalent to that of SVM, reaching an F1-score of 0.5455 and even matching its recall at 0.8571. Although MLP achieved the highest overall accuracy (0.9023), it did not achieve a good F1-score compared to Logistic Regression. Therefore, high accuracy does not always indicate good overall classification quality in terms of the balance between precision and recall.

For the JM1 dataset, all evaluated models presented the lowest performance values. However, Logistic Regression, with an F1 value of 0.35, remained very close to the best black-box model, which was SVM with an F1 value of 0.38.

In the KC1 and KC2 datasets, naturally explainable models outperformed black-box models. In KC1, Logistic Regression achieved an F1-score of 0.3680, while in KC2 it reached 0.708, both higher than those obtained by more complex models.

In the PC1 dataset did Logistic Regression achieve the best overall F1-score (0.395), followed by the black-box models Random Forest (0.375) and Gradient Boosting (0.343), surpassing the remaining naturally explainable models. However, among black-box models, no single model consistently outperformed others; rather, Random Forest and SVM stood out depending on the dataset.

Statistically, Ref. [70] shows that, across all datasets and all predictive metrics, the models exhibit significant differences (either in their means or medians). In [72], it can be observed that, for CMI, MLP is statistically the most accurate model (in terms of both accuracy and precision). For JM1, in almost all metrics the top-ranked model is statistically different from the second-best; however, the best model varies with the metric, with Random Forest, MLP, or Naive Bayes alternately achieving the highest performance. In KC1 and KC2, only one metric shows a statistically significant difference between the best and second-best models, implying that, from a statistical standpoint, using either of the top two models is practically equivalent. Finally, for PC1, Random Forest stands out, although it is statistically superior only for precision and F1.

7.3 Analysis of Explanations by XAI Approach

7.3.1 LIME

Across all five datasets and four black-box models, LIME generated local explanations characterized by selecting only a reduced subset of variables, approximately half of those available, reflected in its complexity value of 0.476. This characteristic remains constant regardless of the model or dataset. Recurring influential variables included `loc`, `iv(g)`, `ev(g)`, `branchCount`, `uniq_Op`, and `uniq_Opnd`.

7.3.2 SHAP

Unlike LIME, SHAP calculates the contribution of each variable considering all possible combinations, resulting in a complexity of 1.0. The most influential variables largely coincide with those observed in LIME. SHAP reveals that variables such as `loc` and `uniq_Opnd` maintain relatively consistent contribution directions across multiple modules.

7.3.3 Breakdown

Breakdown builds explanations sequentially, accumulating the effect of each variable from a baseline intercept. A consistent pattern is the presence of the “+all other factors” term, which in many instances concentrates a contribution equal to or greater than individual influential variables. MLP exhibited more dispersed behavior, reflected in its lower stability (0.76 across all datasets). SVM in JM1 and KC1 showed consistent explanations with stability of 1.0.

7.4 Comparison between LIME, SHAP, and Breakdown

All three techniques consistently identify the same families of variables as most influential in defect prediction: size metrics (`loc`, `b`), cyclomatic complexity metrics (`v(g)`, `iv(g)`, `branchCount`), Halstead metrics (`IOBlank`, `uniq_Op`, `uniq_Opnd`), and documentation metrics (`locCodeAndComment`, `IOComment`) to a lesser extent.

LIME is the most concise but exhibits variability in stability. SHAP provides complete coverage but can hinder rapid interpretation. Breakdown offers a cumulative perspective, transparently showing unexplained portions through the “+all other factors” term.

Finally, given the results in [71], although the non-explainable algorithms exhibit statistically significant differences in terms of stability and interpretability, Table 17 shows that there are no statistically significant differences between the explanation methods themselves.

7.5 Relationship between Observed Explanations and Explainability Metrics

Fidelity of 1.0 for Random Forest, Gradient Boosting, and MLP indicates that post-hoc techniques exactly replicate model behavior. SVM's lower fidelity (0.8 in CM1 and PC1) is reflected in Breakdown, where the “+all other factors” term reaches high values. Stability depends primarily on the base model. Complexity is the only metric that clearly differentiates between post-hoc techniques. The composite interpretability score shows Random Forest and Gradient Boosting achieving the highest values (0.92), MLP maintaining intermediate values (0.88), and SVM exhibiting greater variability (0.78 to 0.9).

Two notable observations emerge from the fidelity and complexity analyses. First, SVM exhibited lower fidelity (0.8) compared to other models (1.0). While this indicates 80% agreement between the surrogate and the original model, the global accuracy-based metric does not explain the sources of disagreement. We attribute this to the highly non-linear decision boundaries induced by the RBF kernel of SVM, which are poorly approximated by linear surrogates, unlike the piecewise constant or linear surfaces of tree-based models and logistic regression. Second, the complexity of LIME remained constant (0.476) across all models. This occurs because LIME returns only the top 10 most relevant features, whereas SHAP and BreakDown assign importance to all input variables. Consequently, the complexity metric reflects an algorithmic artifact rather than a data-driven variation. These findings underscore the need for caution when interpreting aggregate XAI metrics and motivate the use of instance level evaluation in future studies.

7.6 Is it Worthwhile to Apply Post-Hoc Techniques When a Naturally Explainable Model Is Equally Competitive?

Naturally explainable models offer an important advantage: their interpretability is inherent to their structure. Results show that in several datasets, especially KC1 and KC2, Logistic Regression not only competes with but surpasses black-box models in F1-score. In these cases, using more complex models with post-hoc techniques does not provide a clear performance improvement but does introduce methodological complexity and the need to validate explanations.

In contrast, for datasets such as PC1 and JM1, black-box models offer good predictive performance, and explainability techniques yield explanations with acceptable levels of fidelity and stability. In these cases, the use of Explainable AI is justified.

The decision between using naturally explainable models vs. black-box models with post-hoc techniques should not be based solely on predictive performance. It must also consider whether the improvement in accuracy sufficiently compensates for the loss of direct interpretability and the additional effort required to validate explanation quality.

7.7 Practical Implications for Software Practitioners

Our findings provide actionable guidance for software development teams and quality assurance practitioners considering the adoption of XAI-enhanced defect prediction.

First, naturally explainable models should be used as a baseline in any SDP project. Logistic Regression achieved competitive or superior performance in 40% of the datasets (KC1 and KC2), with F1-scores of 0.3677 and 0.7077 respectively, surpassing all black-box models. Practitioners should not automatically default to complex models; a practical workflow is:

- train a naturally explainable model as a baseline;
- if performance is satisfactory and interpretability is a priority, deploy it directly;
- if performance is insufficient, explore black-box models with post-hoc XAI.

Second, the choice of XAI technique should depend on the operational context. Based on our analysis:

- SHAP is recommended when theoretical rigor and completeness are priorities (e.g., regulatory compliance, auditing), despite higher computational cost.
- LIME is preferable when conciseness and human-readability are needed (e.g., non-technical stakeholders), as it selects approximately half of the features (complexity of 0.476).
- Breakdown is suitable for preliminary analyses or resource-constrained environments due to its computational efficiency.

Third, model selection should consider explanation stability. SVM explanations exhibited high variability across cross-validation folds, while Random Forest and Gradient Boosting showed consistent interpretability (0.92). If interpretability is critical for decision-making (e.g., prioritizing modules for testing), ensemble methods are preferable over SVM.

Finally, for researchers, these interpretability metrics (fidelity, stability, and complexity) provide a baseline for future XAI-SDP studies. The finding that Logistic Regression outperformed black-box models in two datasets challenges the assumption that complex models are always better, opening new research questions about when naturally explainable models are preferable.

8 Threats to Validity

8.1 Internal Validity

A limitation of this study that the datasets used (CM1, JM1, KC1, KC2, and PC1) come from the public NASA PROMISE repository, widely used in software defect prediction studies. However, to prepare the data, modules with missing values were directly removed. This may introduce bias if those data were not missing completely at random, as certain types of modules could be systematically eliminated.

Furthermore, all datasets exhibit class imbalance (i.e., there are more modules of one type than another), which may affect the performance of some models. In particular, metrics such as recall may be influenced, as they depend on how the classes are distributed.

8.2 External Validity

The results obtained are limited to the selected datasets, which correspond to embedded software projects developed by NASA and written in the C language. Several important limitations affect the generalizability of the findings. First, the PROMISE datasets are relatively old, collected in the late 1990s and early 2000s, and may not reflect modern software development practices such as continuous integration, microservices, or cloud-native architectures.

Second, the metrics available in PROMISE datasets are limited to static code metrics and do not include process metrics (e.g., code churn, developer activity, commit history, or bug report data), which have been shown to improve defect prediction performance in contemporary studies [12]. Third, all datasets originate from NASA projects, which may not be representative of other domains such as web applications, mobile

development, or enterprise systems. For these reasons, it cannot be assured that the same results would hold in other contexts, such as enterprise software, open-source projects, or systems developed in other programming languages.

8.3 Construct Validity

Several limitations relate to the definition and measurement of interpretability metrics and model performance. First, the metrics used to evaluate explanations (fidelity and stability) were defined based on measures such as variance and cross-validation error. Although this is a valid way to measure interpretability within the functionally-grounded framework proposed by Doshi-Velez and Kim [64], it is not the only one, and there is no universal consensus on how this concept should be evaluated. The functionally-grounded metrics serve as proxies for human interpretability, but they have not been validated through human-subject studies. Prior research has shown that functionally-grounded metrics can be optimized without improving explanation quality. Therefore, it cannot be concluded that high fidelity or stability scores necessarily correspond to explanations that are useful or trustworthy to actual software practitioners. Future work should conduct human-grounded and application-grounded evaluations where developers rate the usefulness and clarity of explanations.

Second, it was decided not to include the Complexity metric within the composite interpretability score calculation. Although this decision has a theoretical justification—based on the additive nature of LIME and SHAP, where the explanatory structure does not change qualitatively between runs—it remains a design choice that could affect comparison with other studies that do include it. To mitigate this, complexity separately as a descriptive characteristic is reported, allowing readers to compare the conciseness of each XAI technique.

Third, the explanations generated by LIME have a random component, as they are based on local sampling. This means that results may vary slightly between executions. To reduce this effect, a fixed random seed was used, which helps maintain result consistency. However, variability across different random seeds was not systematically quantified.

9 Conclusions and Future Work

In this study, a comparative empirical evaluation of seven machine learning algorithms for software defect prediction was conducted, distinguishing between three naturally explainable models (Decision Tree, Logistic Regression, and Naïve Bayes) and four black-box models (Random Forest, Gradient Boosting, Support Vector Machine, and Multilayer Perceptron). The analysis was performed on five publicly available datasets from the NASA PROMISE repository, namely CM1, JM1, KC1, KC2, and PC1, using five-fold stratified cross-validation and SMOTE to address class imbalance. Furthermore, three post-hoc Explainable Artificial Intelligence techniques (LIME, SHAP, and Breakdown) were applied to the black-box models and evaluated the quality of their explanations using fidelity, stability, complexity, and a composite interpretability score.

Our findings reveal that black-box models do not consistently outperform naturally explainable models across all datasets. While black-box models achieved superior predictive performance in JM1, the results were less conclusive in the remaining datasets. In CM1, Logistic Regression tied with SVM (both 0.5455 F1), demonstrating that an explainable model can match the performance of a black-box model. In KC1, KC2, and PC1, naturally explainable Logistic Regression obtained the highest F1-scores, surpassing all black-box models in those datasets. Notably, Random Forest and Gradient Boosting were among the most accurate models overall, although their F1-scores did not always reflect the same level of performance. These results demonstrate that interpretability does not necessarily come at the cost of predictive accuracy and that

naturally explainable models remain a viable and competitive choice in certain software defect prediction contexts. Additionally, it was observed that high accuracy alone is insufficient to evaluate model performance, as MLP achieved the highest accuracy in CMI but a lower F1-score compared to Logistic Regression and SVM, highlighting the importance of precision-recall balance, particularly in imbalanced datasets.

Regarding the evaluation of post-hoc explanations, it was found that fidelity was perfect for Random Forest, Gradient Boosting, and MLP across all datasets and XAI techniques, indicating that LIME, SHAP, and Breakdown faithfully replicate the behavior of these models. However, SVM exhibited lower fidelity in CMI and PCI, limiting the reliability of its explanations in those datasets. Stability varied primarily as a function of the base model rather than the XAI technique, with Random Forest and Gradient Boosting showing consistent stability values, while SVM displayed the highest variability. Complexity was the only metric that clearly distinguished between the three XAI techniques: LIME consistently selected approximately half of the available variables, producing concise and human-readable explanations, whereas SHAP and Breakdown included all variables, offering complete but potentially overwhelming explanations. Finally, the composite interpretability score, which combines fidelity and stability, showed that Random Forest and Gradient Boosting achieved the highest scores, indicating that their explanations are both faithful and consistent, whereas SVM exhibited a wider range, suggesting that the reliability of its explanations depends heavily on the dataset.

Overall, it is concluded that while black-box models generally offer superior predictive power in certain software defect prediction contexts, naturally explainable models remain competitive and should not be overlooked. The decision to adopt black-box models with post-hoc XAI should be guided not only by predictive performance but also by the quality of the generated explanations as measured by fidelity, stability, and complexity.

In reflecting on the scope of this study, we acknowledge that our contribution is primarily incremental. Existing XAI techniques and standard machine learning models into a systematic framework were integrated, rather than introducing fundamentally new algorithms. The value of our work lies in the rigorous empirical comparison, the proposal of interpretability metrics tailored to SDP, and the provision of actionable guidance for practitioners. Nevertheless, it is recognized several limitations: our metrics require human-subject validation, our datasets are relatively old and may not generalize to modern software systems, and our metric design remains relatively simple. We therefore view this work as a foundational step one that establishes a baseline for future research on XAI in SDP, including human-centered evaluation, validation on contemporary datasets, and the development of more sophisticated interpretability metrics.

Several directions for future research emerge from this study. First, the current experiment was limited to five datasets from the NASA PROMISE repository, which correspond to embedded software projects written in C. To enhance the external validity and generalizability of our findings, future work should replicate the experimental design on a broader range of datasets, including open-source projects such as Eclipse or Apache, enterprise software systems, and projects developed in other programming languages such as Java, Python, or C++.

Second, while this study evaluated LIME, SHAP, and Breakdown, emerging XAI techniques such as Anchors, Contrastive Explanations, and Integrated Gradients warrant investigation. A comparative analysis across a wider spectrum of post-hoc methods could help identify which techniques best balance fidelity, stability, and complexity specifically within the software defect prediction domain.

Third, the composite interpretability score proposed in this work combines fidelity and stability. Future research could explore alternative formulations, including weighted averages or non-linear combinations,

and investigate whether additional dimensions—such as user satisfaction, task completion time, or mental effort—should be incorporated into interpretability evaluation frameworks.

Finally, beyond applying post-hoc XAI to black-box models, future research could focus on designing inherently interpretable models that achieve competitive predictive performance by incorporating domain-specific constraints or regularization strategies inspired by software defect prediction knowledge. Additionally, the development of interactive visualization tools that present LIME, SHAP, and Break-down explanations in user-friendly interfaces, followed by empirical user studies, would help evaluate how different explanation formats impact trust, comprehension, and decision quality among software engineering practitioners.

These future directions aim to advance the integration of XAI into SDP, ultimately fostering more transparent, trustworthy, and actionable predictive models for the software engineering community.

Acknowledgement: The authors would like to thank the Facultad de Estadística e Informática, Universidad Veracruzana, México.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Miriam Ramírez-Zárate: Conceptualization, Software, Investigation, Data curation, Writing—original draft. Ángel J. Sánchez-García: Conceptualization, Methodology, Formal analysis, Validation, Writing—original draft, Supervision, Project administration. Lizbeth A. Hernández-González: Conceptualization, Supervision, Validation, Writing—review & editing Saúl Domínguez-Isidro: Conceptualization, Visualization, software, Writing—review & editing. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available in Zenodo [67–72].

Ethics Approval: This study did not involve human participants, animal subjects, or sensitive personal data. All experiments were conducted using publicly available benchmark datasets.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pressman RS. Software engineering: a practitioner's approach. 7th ed. New York, NY, USA: McGraw-Hill; 2009.
2. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng.* 2007;33(1):2–13. doi:10.1109/tse.2007.256941.
3. IEEE Computer Society. IEEE standard for software quality assurance processes. New York, NY, USA: IEEE; 2014.
4. Olivares-Galindo JA, Sánchez-García AJ, Barrientos-Martínez RE, Ocharán-Hernández JO. Ensemble classifiers in software defect prediction: a systematic literature review. In: Proceedings of the 2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT); 2023 Nov 6–10; León, Mexico. Washington, DC, USA: IEEE Computer Society. p. 1–8.
5. Dietterich TG. Ensemble methods in machine learning. In: International workshop on multiple classifier systems. Cham, Switzerland: Springer; 2000. p. 1–15.
6. Molnar C. Interpretable machine learning. Lulu.com; 2022 [cited 2026 Jan 1]. Available from: https://books.google.com/books?hl=es&lr=&id=jBm3DwAAQBAJ&oi=fnd&pg=PP1&dq=CChristoph+Molnar+Interpretable+machine+learning.+Lulu.com,+2022&ots=EhzW_kBGQ3&sig=PpLz_PX4SBnghUMCglI2xI4eiAg.
7. Ribeiro MT, Singh S, Guestrin C. “Why should i trust you?” Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016 Aug 13–17; San Francisco, CA, USA. p. 1135–44.

8. Kim S, Whitehead EJ, Zhang Y. Classifying software changes: clean or buggy? *IEEE Trans Softw Eng.* 2008;34(2):181–96.
9. Jiang T, Tan L, Kim S. Personalized defect prediction. In: *Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*; 2013 Nov 11–15; Palo Alto, CA, USA. New York, NY, USA: IEEE; 2013. p. 279–89.
10. Wang S, Wang J, Nam J, Nagappan N. Continuous software bug prediction. In: *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*; 2021 Oct 11–15; Bari, Italy. p. 1–12.
11. Herbold S. On the costs and profit of software defect prediction. *IEEE Trans Softw Eng.* 2019;47(11):2617–31. doi:10.1109/tse.2019.2957794.
12. Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans Softw Eng.* 2011;38(6):1276–304. doi:10.1109/tse.2011.103.
13. Hemández-Molinos MJ, Sánchez-García ÁJ, Barrientos-Martínez RE. Classification algorithms for software defect prediction: a systematic literature review. In: *Proceedings of the 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*; 2021 Oct 25–29; San Diego, CA, USA. New York, NY, USA: IEEE; 2021. p. 189–96.
14. Matloob F, Ghazal TM, Taleb N, Aftab S, Ahmad M, Khan MA, et al. Software defect prediction using ensemble learning: a systematic literature review. *IEEE Access.* 2021;9:98754–71. doi:10.1109/access.2021.3095559.
15. Pachouly J, Ahirrao S, Kotecha K, Selvachandran G, Abraham A. A systematic literature review on software defect prediction using artificial intelligence: datasets, data validation methods, approaches, and tools. *Eng Appl Artif Intell.* 2022;111(12):104773. doi:10.1016/j.engappai.2022.104773.
16. Khan MA, Elmitwally NS, Abbas S, Aftab S, Ahmad M, Fayaz M, et al. Software defect prediction using artificial neural networks: a systematic literature review. *Sci Program.* 2022;2022(1):2117339. doi:10.1155/2022/2117339.
17. Cedeño GON, Moya KC, Dormond AS, González-Torres A, Rojas-Hernández Y. Systematic literature review: machine learning for software fault prediction. In: *Proceedings of the 2023 IEEE 41st Central America and Panama Convention (CONCAPAN XLI)*; 2023 Nov 8–10; Tegucigalpa, Honduras. New York, NY, USA: IEEE; 2023. p. 1–6.
18. Chengxiao Y, Owais J, Ahmed A, Khan MO, Xiaoyang Z, Tunio MH. Software defect prediction using machine learning—a systematic literature review. In: *Proceedings of the 2023 20th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*; 2023 Dec 15–17; Chengdu, China. New York, NY, USA: IEEE; 2023. p. 1–9.
19. Gezici B, Tarhan AK. Explainable AI for software defect prediction with gradient boosting classifier. In: *Proceedings of the 2022 7th International Conference on Computer Science and Engineering (UBMK)*; 2022 Sep 14–16; Diyarbakir, Turkey. New York, NY, USA: IEEE; 2022. p. 1–6.
20. Bommi NS, Negi A. A standard baseline for software defect prediction: using machine learning and explainable AI. In: *Proceedings of the 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*; 2023 Jun 26–30; Torino, Italy. New York, NY, USA: IEEE; 2023. p. 1798–803.
21. Asal B, Demir MÖ. Enhancing software defect prediction through explainable AI: integrating SHAP and LIME in a voting classifier framework. In: *Proceedings of the 2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP)*; 2024 Sep 21–22; Malatya, Turkey. New York, NY, USA: IEEE; 2024. p. 1–7.
22. Begum M, Shuvo MH, Ashraf I, Al Mamun A, Uddin J, Samad MA. Software defects identification: results using machine learning and explainable artificial intelligence techniques. *IEEE Access.* 2023;11:132750–65.
23. Shin J, Aleithan R, Nam J, Wang J, Harzevili NS, Wang S. An empirical study on the stability of explainable software defect prediction. In: *Proceedings of the 2023 30th Asia-Pacific Software Engineering Conference (APSEC)*; 2023 Dec 4–7; Seoul, Republic of Korea. New York, NY, USA: IEEE; 2023. p. 141–50.
24. Mohapatra S, Ju E, Lee J, Ryu D. Enhancing software defect prediction: integrating SAINT with ant colony optimization and explainable AI. In: *Proceedings of the 2025 IEEE International Conference on Big Data and Smart Computing (BigComp)*; 2025 Feb 9–12; Kota Kinabalu, Malaysia. New York, NY, USA: IEEE; 2025. p. 113–4.
25. Begum M, Shuvo MH, Nasir MK, Hossain A, Hossain MJ, Ashraf I, et al. LCNN: lightweight CNN architecture for software defect feature identification using Explainable AI. *IEEE Access.* 2024;12:55744–56.

26. Lee G, Lee SUJ. An empirical comparison of model-agnostic techniques for defect prediction models. In: Proceedings of the 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER); 2023 Mar 21–24; Macao, China. New York, NY, USA: IEEE; 2023. p. 179–89.
27. Lee G, Ju H, Lee SUJ. Can we trust the actionable guidance from explainable AI techniques in defect prediction? In: Proceedings of the 2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER); 2025 Mar 4–7; Montreal, QC, Canada. New York, NY, USA: IEEE; 2025. p. 476–87.
28. Halder S, Capretz LF. Explainable software defect prediction from cross company project metrics using machine learning. In: Proceedings of the 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS); 2023 May 17–19; Madurai, India. New York, NY, USA: IEEE; 2023. p. 150–7.
29. Khan A, Ali A, Khan J, Ullah F, Faheem M. Exploring consistent feature selection for software fault prediction: an XAI-based model-agnostic approach. *IEEE Access*. 2025;13(4):75493–524. doi:10.1109/access.2025.3558913.
30. Santos G, Figueiredo E, Veloso A, Vigiato M, Ziviani N. Predicting software defects with explainable machine learning. In: Proceedings of the XIX Brazilian Symposium on Software Quality; 2020 Dec 1–4; São Luís, Brazil. p. 1–10.
31. Santos G, Veloso A, Figueiredo E. Understanding thresholds of software features for defect prediction. In: Proceedings of the XXXVI Brazilian Symposium on Software Engineering; 2022 Oct 5–7; Virtual. p. 305–10. doi:10.1145/3555228.3555269.
32. Zivkovic T, Nikolic B, Simic V, Pamucar D, Bacanin N. Software defects prediction by metaheuristics tuned extreme gradient boosting and analysis based on shapley additive explanations. *Appl Soft Comput*. 2023;146(1):110659. doi:10.1016/j.asoc.2023.110659.
33. Villoth JP, Zivkovic M, Zivkovic T, Abdel-salam M, Hammad M, Jovanovic L, et al. Two-tier deep and machine learning approach optimized by adaptive multi-population firefly algorithm for software defects prediction. *Neurocomputing*. 2025;630(7):129695. doi:10.1016/j.neucom.2025.129695.
34. Al-Smadi Y, Eshay M, Al-Qerem A, Nashwan S, Ouda O, Abd El-Aziz A. Reliable prediction of software defects using Shapley interpretable machine learning models. *Egypt Inform J*. 2023;24(3):100386. doi:10.1016/j.eij.2023.05.011.
35. Yang F, Zeng G, Zhong F, Zheng W, Xiao P. Interpretable software defect prediction incorporating multiple rules. In: Proceedings of the 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER); 2023 Mar 21–24; Macao, China. New York, NY, USA: IEEE; 2023. p. 940–7.
36. Villoth SJ, Villoth JP, Zivkovic T, Zivkovic M, Jovanovic L, Bacanin N, et al. Adaboost optimized by sinh cosh algorithm for prediction of software defects. In: International Conference on Intelligent Systems and Pattern Recognition. Cham, Switzerland: Springer; 2024. p. 53–68.
37. Bacanin N, Kostic V, Dragić R, Jovanovic L, Zivkovic M, Radomirovic B, et al. Applied modified metaheuristic: machine learning tuned for software defect detection. In: International Conference on Advancements in Smart Computing and Information Security. Cham, Switzerland: Springer; 2025. p. 120–30.
38. Malhotra R. Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Appl Soft Comput*. 2014;21(1):286–97. doi:10.1016/j.asoc.2014.03.032.
39. Shirabad JS, Menzies TJ. The PROMISE repository of software engineering databases. School of information technology and engineering, University of Ottawa, Canada. 2005 [cited 2026 Jan 1]. Available from: <https://cir.nii.ac.jp/crid/1370013170018460421>.
40. Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Trans Softw Eng*. 2008;34(4):485–96.
41. Little RJ, Rubin DB. Statistical analysis with missing data. New York, NY, USA: John Wiley & Sons; 2019.
42. Kotsiantis SB, Zaharakis I, Pintelas P. Supervised machine learning: a review of classification techniques. *Emerg Artif Intell Appl Comput Eng*. 2007;160(1):3–24.
43. Han J, Kamber M, Pei J. Data mining: concepts and techniques. Waltham, MA, USA: Morgan Kaufmann Publishers; 2012. Available from: <https://www.sciencedirect.com/book/monograph/9780123814791/data-mining-concepts-and-techniques>.

44. Lundberg SM, Lee SI. A unified approach to interpreting model predictions. *Adv Neural Inf Process Syst*. 2017;30:4765–74. doi:10.48550/arxiv.1705.07874.
45. Staniak M, Biecek P. Explanations of model predictions with live and breakDown packages. arXiv:1804.01955. 2018.
46. Guidotti R, Monreale A, Ruggieri S, Turini F, Giannotti F, Pedreschi D. A survey of methods for explaining black box models. *ACM Comput Surv (CSUR)*. 2018;51(5):1–42. doi:10.1145/3236009.
47. Lundberg SM, Erion GG, Lee SI. Consistent individualized feature attribution for tree ensembles. arXiv:1802.03888. 2018.
48. Biecek P, Burzykowski T. Explanatory model analysis: explore, explain, and examine predictive models. Boca Raton, FL, USA: Chapman and Hall/CRC; 2021.
49. Hastie T, Tibshirani R, Friedman J. The elements of statistical learning. Springer series in statistics New-York. New York, NY, USA: Springer; 2009.
50. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. *J Artif Intell Res*. 2002;16:321–57.
51. Lipton ZC. The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery. *Queue*. 2018;16(3):31–57. doi:10.1145/3236386.3241340.
52. Breiman L, Friedman J, Olshen RA, Stone CJ. Classification and regression trees. Boca Raton, FL, USA: Chapman and Hall/CRC; 2017.
53. Quinlan JR. C4. 5: Programs for machine learning. Amsterdam, The Netherlands: Elsevier; 2014.
54. Friedman N, Geiger D, Goldszmidt M. Bayesian network classifiers. *Mach Learn*. 1997;29(2):131–63. doi:10.1023/a:1007465528199.
55. Hosmer Jr DW, Lemeshow S, Sturdivant RX. Applied logistic regression. New York, NY, USA: John Wiley & Sons; 2013.
56. Breiman L. Random forests. *Mach Learn*. 2001;45(1):5–32. doi:10.1023/a:1010933404324.
57. Cánovas-García F, Alonso-Sarría F, Gomariz-Castillo F, Oñate-Valdivieso F. Modification of the random forest algorithm to avoid statistical dependence problems when classifying remote sensing imagery. *Comput Geosci*. 2017;103(3):1–11. doi:10.1016/j.cageo.2017.02.012.
58. Goodfellow I, Bengio Y, Courville A. Deep learning, Vol. 1. Cambridge, MA, USA: MIT Press; 2016.
59. Powers DM. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. arXiv:2010.16061. 2020.
60. Sokolova M, Lapalme G. A systematic analysis of performance measures for classification tasks. *Inf Process Manag*. 2009;45(4):427–37.
61. Fawcett T. An introduction to ROC analysis. *Pattern Recognit Lett*. 2006;27(8):861–74. doi:10.1016/j.patrec.2005.10.010.
62. Chawla NV, Japkowicz N, Kotcz A. Special issue on learning from imbalanced data sets. *ACM SIGKDD Explor Newsl*. 2004;6(1):1–6.
63. Saito T, Rehmsmeier M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS One*. 2015;10(3):e0118432. doi:10.1371/journal.pone.0118432.
64. Doshi-Velez F, Kim B. Towards a rigorous science of interpretable machine learning. arXiv:1702.08608. 2017.
65. Adadi A, Berrada M. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE Access*. 2018;6:52138–60.
66. Alvarez-Melis D, Jaakkola TS. Towards robust interpretability with self-explaining neural networks. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*; 2018 Dec 2–8; Montreal, QC, Canada. p. 7786–95.
67. Sánchez-García AJ. LIME results. Zenodo. 2026. doi:10.5281/zenodo.19967320.
68. Sánchez-García AJ. SHAP results. Zenodo. 2026. doi:10.5281/zenodo.19967496.
69. Sánchez-García AJ. Breakdown results. Zenodo. 2026. doi:10.5281/zenodo.19967840.
70. Sánchez-García AJ. Normal statistical test for accuracy. Zenodo. 2026. doi:10.5281/zenodo.21699240.
71. Sánchez-García AJ. Normal statistical test for interpretability. Zenodo. 2026. doi:10.5281/zenodo.21699312.
72. Sánchez-García AJ. Accuracy paired tests. Zenodo. 2026. doi:10.5281/zenodo.21699297.