A Systematic Literature Review on Explainability for ML/DL-based Software Engineering

SICONG CAO, Yangzhou University, Yangzhou, China
XIAOBING SUN, Yangzhou University, Yangzhou, China
RATNADIRA WIDYASARI, Singapore Management University, Singapore, Singapore
DAVID LO, Singapore Management University, Singapore, Singapore
XIAOXUE WU, Yangzhou University, Yangzhou, China
LILI BO, Yangzhou University, Yangzhou, China
JIALE ZHANG, Yangzhou University, Yangzhou, China
BIN LI, Yangzhou University, Yangzhou, China
WEI LIU, Yangzhou University, Yangzhou, China
DI WU, University of Southern Queensland, Toowoomba, Australia
YIXIN CHEN, Washington University in St Louis, St Louis, United States

The remarkable achievements of Artificial Intelligence (AI) algorithms, particularly in Machine Learning (ML) and Deep Learning (DL), have fueled their extensive deployment across multiple sectors, including Software Engineering (SE). However, due to their black-box nature, these promising AI-driven SE models are still far from being deployed in practice. This lack of explainability poses unwanted risks for their applications in critical tasks, such as vulnerability detection, where decision-making transparency is of paramount importance. This article endeavors to elucidate this interdisciplinary domain by presenting a systematic literature review of approaches that aim to improve the explainability of AI models within the context of SE. The review canvasses work appearing in the most prominent SE and AI conferences and journals, and spans 108 articles across 23 unique SE tasks. Based on three key Research Questions (RQs), we aim to (1) summarize the SE tasks

This work is supported by the National Natural Science Foundation of China (No. 62572421, No. 62002309, No. 62202414); China Postdoctoral Science Foundation (No. 2025T180438); the Jiangsu "333" Project; the Open Project of Industry-Education Integration Innovation Center of Specialized Cybersecurity (Yangzhou University, No. YZUCSC2025KF02); the Open Foundation of Yunnan Key Laboratory of Software Engineering (No. 2023SE201), and the National Research Foundation, under its Investigatorship Grant (NRF-NRFI08-2022-0002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.
Authors' Contact Information: Sicong Cao, Yangzhou University, Yangzhou, Jiangsu, China and Industry-Education Integration Innovation Center of Specialized Cybersecurity (Yangzhou University), Yangzhou, Jiangsu, China; e-mail: DX120210088@yzu.edu.cn; Xiaobing Sun (corresponding author), Yangzhou University, Yangzhou, Jiangsu, China; e-mail: xbsun@yzu.edu.cn; Ratnadira Widyasari, Singapore Management University, Singapore, Singapore, Singapore; e-mail: ratnadiraw.2020@phdcs.smu.edu.sg; David Lo, Singapore Management University, Singapore, Singapore; e-mail: davidlo@smu.edu.sg; Xiaoxue Wu, Yangzhou University, Yangzhou, Jiangsu, China; e-mail: xiaoxuewu@yzu.edu.cn; Lili Bo, Yangzhou University, Yangzhou, Jiangsu, China and Yunnan Key Laboratory of Software Engineering, Kunming, Yunnan, China; e-mail: lilibo@yzu.edu.cn; Jiale Zhang, Yangzhou University, Yangzhou, Jiangsu, China; e-mail: jialezhang@yzu.edu.cn; Bin Li, Yangzhou University, Yangzhou, Jiangsu, China; e-mail: lb@yzu.edu.cn; Wei Liu, Yangzhou University, Yangzhou, Jiangsu, China; e-mail: weiliu@yzu.edu.cn; Di Wu, University of Southern Queensland, Toowoomba, Queensland, Australia; e-mail: di.wu@unisq.edu.au; Yixin Chen, Washington University in St Louis, St Louis, Missouri, United States; e-mail: chen@cse.wustl.edu.

ACM 0360-0300/2025/10-ART95
https://doi.org/10.1145/3763230

where XAI techniques have shown success to date; (2) classify and analyze different XAI techniques; and (3) investigate existing evaluation approaches. Based on our findings, we identified a set of challenges remaining to be addressed in existing studies, together with a set of guidelines highlighting potential opportunities we deemed appropriate and important for future work.

CCS Concepts: • **General and reference** → *Surveys and overviews*; • **Computing methodologies** → *Neural networks*; *Artificial intelligence*; • **Software and its engineering** → *Software development techniques;*

Additional Key Words and Phrases: Explainable AI, XAI, interpretability, neural networks, survey

ACM Reference Format:
Sicong Cao, Xiaobing Sun, Ratnadira Widyasari, David Lo, Xiaoxue Wu, Lili Bo, Jiale Zhang, Bin Li, Wei Liu, Di Wu, and Yixin Chen. 2025. A Systematic Literature Review on Explainability for ML/DL-based Software Engineering. *ACM Comput. Surv.* 58, 4, Article 95 (October 2025), 34 pages. https://doi.org/10.1145/3763230

1 Introduction

Software Engineering (SE) is a discipline that deals with the design, development, testing, and maintenance of software systems. As software continues to pervade a wide range of industries, diverse and complex SE data, such as source code, bug reports, and test cases, have grown to become unprecedentedly large and complex. Driven by the success of **Artificial Intelligence (AI)** algorithms in various research fields, the SE community has shown great enthusiasm for exploring and applying advanced **Machine Learning (ML)/Deep Learning (DL)** models to automate or enhance SE tasks typically performed manually by developers, including automated program repair [50, 128], code generation [70], and vulnerability detection [8, 176]. A recent report from the 2021 SEI Educator's Workshop has referred to **AI for Software Engineering (AI4SE)** as an umbrella term to describe research that uses AI algorithms to tackle SE tasks [95].

Despite the unprecedented performance achieved by ML/DL models with higher complexity, they have been slow to be deployed in the SE industry. This reluctance arises due to prioritizing accuracy over **Explainability**—AI systems are notoriously difficult to understand for humans because of their complex configurations and large model sizes [42]. From the perspective of the model user, explainability is needed to establish trust when imperfect *"black-box"* models are used. For instance, a developer may seek to comprehend the rationale behind a DL-based vulnerability detection model's decision, i.e., why it predicts a particular code snippet as vulnerable, to facilitate analyzing and fixing the vulnerability [93, 166]. For the model designers, explainability is required to investigate failure cases and direct the weak AI models in the proper paths as intended [127]. In other words, merely a simple decision result (e.g., a binary classification label) without any explanation is often not good enough. This fact stimulates the urgent demand for designing algorithms capable of explaining the decision-making process of black-box AI models, leading to the creation of a novel research topic termed **eXplainable AI (XAI)** [29].

Our Work. To effectively chart the most promising path forward for research on the explainability for AI4SE, we conducted a **Systematic Literature Review (SLR)** to bridge this gap, providing valuable insights to the community. In particular, we followed the standardized practice suggested by Zhang et al. [165], which included three main steps: (❶) designing search strategies; (❷) study selection; and (❸) data extraction and analysis. Due to space limitations, we detailed each step of our SLR and results online as supplementary materials.[1] As a result, we examined 108 primary studies published in 27 flagship conferences and journals over the last 13 years (2012–2024). In this article, we focused on investigating the following **Research Questions (RQs)**:

[1] https://github.com/RISS-Vul/xai4se-paper/blob/master/Appendix.pdf

— ***RQ_1: What types of AI4SE studies have been explored for explainability?***
Findings: *(1) Primary studies can be categorized into 23 unique SE tasks across five major activities within software development life cycle; (2) Early studies predominantly concentrated on traditional classification tasks like Bug/Defect Prediction. Since 2021, researchers have turned their attention to a greater number of more complex SE tasks; (3) While there has been a recent wealth of work, there are still underrepresented topics in software requirements and design and software management that should be considered by the SE community, suggesting a potential area of focus for future research in this field.*

— ***RQ_2: How XAI techniques are used to support SE tasks?***
  – ***RQ_{2a}:*** *What types of XAI techniques are employed to generate explanations?*
  – ***RQ_{2b}:*** *What format of explanation is provided for various SE tasks?*
Findings: *(1) Existing XAI techniques for SE tasks are mainly developed along five directions, including* ***Out-of-the-Box Toolkit*** *(**OT**),* ***Interpretable Model*** *(**IM**),* ***Domain Knowledge*** *(**DK**),* ***Attention Mechanism*** *(**AM**), as well as a set of other highly tailored approaches. The most popular of the five being OT, contributing ≈34% of our surveyed studies; (2) Key factors guiding the selection encompass Task Fitness, Model Compatibility, and Stakeholder Preference; (3) A number of explanation formats have been explored in our surveyed studies, with the main formats utilized being Numeric, Text, Visualization, Source Code, and Rule. They were often tightly associated with a given SE task.*

— ***RQ_3: How well do XAI techniques perform in supporting various SE tasks?***
  – ***RQ_{3a}:*** *What baseline techniques are used to evaluate XAI4SE approaches?*
  – ***RQ_{3b}:*** *What benchmarks are used for these comparisons?*
  – ***RQ_{3c}:*** *What evaluation metrics are employed to measure XAI4SE approaches?*
Findings: *(1) In light of a notable scarcity of well-documented and reusable baselines or benchmarks, approximately 28.7% of the benchmarks employed in the evaluations of our studied approaches were self-generated, with a significant portion not being publicly accessible or reusable; (2) There is no consensus on evaluation strategies for XAI4SE studies, and in many cases, the evaluation is only based on specific properties, such as correctness and coherence, or researchers' subjective intuition of what constitutes a good explanation.*

Contributions. We anticipate that our findings will be instrumental in guiding future advancements in this rapidly evolving field. This study makes the following contributions:

— We present a systematic review of recent 108 primary studies on the topic of explainability for ML/DL-based SE, and pinpoint several potential directions for researchers and practitioners.
— We describe the key applications of XAI4SE encompassing a diverse range of 23 unique SE tasks, grouped into five core SE activities within software development life cycle.
— We synthesize a taxonomy of XAI techniques used in SE from an integration perspective, and analyze frequently used formats of explanation.
— We summarize common evaluation means adopted by XAI4SE research, including available baselines, prevalent benchmarks, and commonly employed evaluation metrics, to determine their validity.
— We discuss key challenges that using XAI techniques encounters within the SE field, and provide several practical guidelines for future research.
— We maintain an interactive website, https://riss-vul.github.io/xai4se-paper/, with all of our data and results for reproducibility, and encourage contributions from the community to continue to push forward XAI4SE research.

Comparison with Existing Surveys. Recently, the SE community has embarked on a series of research activities regarding explainability, where several existing literature reviews or surveys

Table 1. Comparison of Our Work with Previous Surveys/Reviews on Explainability for AI4SE Research

Reference	Studied Model	Scope	# Articles	Taxonomy	Evaluation†			Guideline
					Baseline	Benchmark	Metric	
Mohammadkhani et al. [85]	ML and DL	-2022	24	General	○	◐	◐	○
Yang et al. [155]	CodeLMs	2019–2023	146 (16)	General	○	○	○	○
Our survey	ML and DL (+LLM)	2012–2024	108	Customized	●	●	●	●

†○ denotes not cover, ◐ denotes partial cover, and ● denotes fully cover.

[85, 155] have been produced, as summarized in Table 1. Mohammadkhani et al. [85] conducted a seminal survey on explainable AI for SE research. They primarily focus on the explainability of AI4SE models over the *which*, the *what*, and the *how* dimensions, i.e., *which* SE tasks are being explained, *what* types of XAI techniques are adopted, and *how* they are evaluated. Yang et al. [155] reviewed 146 studies to investigate how non-functional properties of **Code Language Models (CodeLMs)**, including robustness, security, privacy, explainability, efficiency, and usability, were evaluated and enhanced. Despite the similarity in terms of the high-level topic, there remain some fundamental differences. First, they either focused narrowly on a single model architecture (e.g., CodeLMs) or only analyzed a small fraction of relevant literature published until June 2022. As a result, insights derived from them may not be generalizable to *all* AI4SE solutions, or may not keep pace with the ongoing development of the community. Second, they directly borrowed the general taxonomy, i.e., ante- and post-hoc explanation, from the XAI field to classify explanation techniques in SE tasks. This taxonomy is coarse-grained and may not be applicable to stakeholders with distinct objectives and expertise. Third, they did not (or only partially) explicitly discuss the evaluation aspects of reviewed articles, including available baselines, prevalent benchmarks, and commonly employed evaluation metrics. The lack of comprehensive evaluation may pose obstacles to readers interested in deploying XAI techniques in practical SE scenarios. Overall, by systematically reviewing publications from 2012 to 2024, spanning 13 years of research, we synthesize a detailed research roadmap of past work on XAI4SE, complete with identified open challenges and best guidelines for applying XAI techniques to SE tasks.

Article Organization. The remainder of this article is structured as follows: Section 2 describes the preliminaries of XAI. The succeeding Sections 3–5 are devoted to answering each of these RQs individually. Section 6 discusses the challenges that still need to be solved and points out potential research opportunities. Section 7 outlines guidelines for conducting future work on XAI4SE based upon the findings of our SLR. Section 8 concludes this article.

2 XAI: Preliminaries

This section first details several critical terminologies commonly used in the XAI field. Then, we offer a general overview of the taxonomy of XAI approaches, aiming to furnish the reader with a solid comprehension of this topic.

2.1 Definition

The greatest challenge in establishing the concept of XAI in SE is the ambiguous definition of *interpretability* and *explainability*. Those terms, together with *interpretation* and *explanation*, are often used interchangeably in the literature [7, 89]. For example, quoting Doshi-Velez and Kim et al. [28], interpretability is the ability "to explain or to present in understandable terms to a human." By contrast, according to Lent et al. [130], an explainable AI means it can "present the user with an easily understood chain of reasoning from the user's order, through the AI's knowledge and inference, to the resulting behavior." Some argue that the terms are closely related but distinguish between them, although there is no consensus on what the distinction exactly is [3, 91]. To ensure that we do not exclude work because of different terminologies, we equate them (and use them interchangeably) to

keep a general, inclusive discussion regardless of this debate. In this survey, we frame explanations in the context of SE research using ML/DL and adopt the phrasing of Dam et al. [20] as follows:

> *Definition 1.* Explainability or Interpretability of an AI-powered SE model measures the degree to which a human observer can understand the reasons behind its decision (e.g. a prediction).

Under this context, there are two distinct ways of achieving explainability: (❶) making the entire decision-making process transparent and comprehensible (i.e., white-box/IMs); and (❷) explicitly providing an explanation for each decision (i.e., surrogate models). In addition, since SE is task-oriented, explanations in SE tasks should be viewed from a perspective that values practical use [146]. As we observed in Section 4.2, there are multiple legitimate types of explanations for SE practitioners who have different intents and expertise.

2.2 Taxonomy

Taxonomy is a useful tool to get an overview of the emerging field. Based on the previous literature [119], most XAI approaches can be categorized according to three criteria: (❶) *scope* (local vs. global); (❷) *stage* (ante-hoc vs. post-hoc); (❸) and *portability* (model-specific vs. model-agnostic), as illustrated in Figure 1.

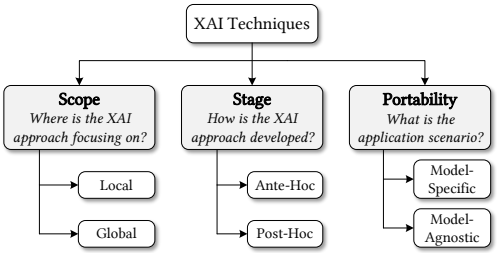


Fig. 1. General taxonomy of the survey in terms of scope, stage, and portability.

Classification by Scope. The scope of explanations can be categorized as either *local* or *global* (some approaches can be extended to both) according to whether the explanations provide insights about the model functioning for the general data distribution or for a specific data sample, respectively. Local explainability approaches, such as LIME [108] and SHAP [78], seek to explain why a model performs a specific prediction for an individual input. Global explainability approaches work on an array of inputs to give insights into the overall behavior of the black-box model. Various rule-based models such as decision trees are in this category.

Classification by Stage. XAI can be categorized based on whether the explanation mechanism is inherent within the model's internal architecture or is implemented following the model's learning/development phase. The former is named *ante-hoc explainability* (also known as *intrinsic explainability* or *self-explainability*), while the latter refers to *post-hoc explainability*. Most inherently interpretable approaches are model-specific such that any change in the architecture will need significant changes in the approach itself. By contrast, post-hoc approaches typically operate by perturbing parts of the data in a high-dimensional vector space to discern the contributions of various features to the model's predictions, or by analytically ascertaining the influence of different features on the prediction outcomes.

Classification by Portability. According to the models they can be applied to, explanation approaches can be further classified as *model-specific* and *model-agnostic*. Model-specific approaches require access to the internal model architecture, meaning that they are restricted to explain only one specific family of models. Conversely, model-agnostic approaches can be used to explain arbitrary models without being constrained to any particular model architecture. Unlike other works that focus on underlying models and architectures, in this SLR, we design our taxonomy

Table 2. Distribution of SE Tasks over Five SE Activities

SE Activity	SE Task (# Articles)	References
Software Requirements and Design (Section 3.1.1)	Requirement Classification (1)	[19]
Software Development (Section 3.1.2)	Code Understanding (11)	[4, 17, 66, 81, 90, 97, 104, 109, 134, 139, 153]
	Program Synthesis (7)	[12, 21, 30, 58, 73, 92, 94]
	Code Summarization (3)	[38, 51, 98]
	Code Search (2)	[133, 135]
	API Recommendation (1)	[47]
Software Testing (Section 3.1.3)	Test Case-Related (5)	[2, 54, 57, 124, 161]
	Debugging (4)	[16, 39, 55, 60]
	Vulnerability Detection (14)	[9, 13, 15, 34, 46, 65, 75, 93, 123, 127, 166, 170, 175, 177]
	Bug/Fault Localization (3)	[56, 143, 144]
Software Maintenance (Section 3.1.4)	Program Repair (2)	[6, 83]
	Malware/Anomaly Detection (12)	[45, 63, 64, 68, 72, 74, 79, 137, 147, 164, 172, 173]
	Bug/Defect Prediction (19)	[11, 26, 36, 37, 52, 53, 61, 69, 82, 87, 102, 103, 105, 117, 140, 151, 152, 162, 174]
	OSS Sustainability Prediction (1)	[149]
	Root Cause Analysis (5)	[24, 67, 106, 125, 156]
	Code Review (2)	[113, 154]
	Code Smell Detection (6)	[48, 107, 129, 136, 138, 159]
	Code Clone Detection (1)	[1]
	Bug Report-Related (5)	[23, 44, 59, 114, 116]
Software Management (Section 3.1.5)	Mining Software Repositories (1)	[71]
	Configuration Extrapolation (1)	[25]
	Effort/Cost Estimation (1)	[35]
	Developer Recommendation (1)	[150]

criteria with greater emphasis on the integration perspective of XAI and SE (Section 4.1), making them better suited to the requirements of the SE community.

3 RQ_1: What Types of AI4SE Studies Have Been Explored for Explainability?

This RQ aims to investigate the application scenarios of XAI techniques in helping improve the explainability of various AI4SE models. In total, we identified 23 separate SE tasks where an XAI technique had been applied. These tasks span across five main phases of **Software Development Life Cycle (SDLC)** [112]. The full taxonomy is displayed in Table 2, which associates the relevant primary study paired with the SE task and activity it belongs to.

3.1 How XAI Are Used in Specific SE Tasks?

In this subsection, we delved into the progress of various SE tasks[2] that applied XAI techniques. By investigating this RQ, we aimed to obtain a clear understanding of what has been done and what else can be done.

3.1.1 SE Tasks in Software Requirements and Design. Software requirements refer to specific descriptions of conditions or capabilities needed by users, systems, or system components, while software design involves the process of defining the structure, components, functionalities, interfaces, and their relationships within a software system. During this phase, only one topic, i.e., *Requirement Classification*, is explored, leaving ample space for further exploration.

Requirement Classification. As a key example of ML applied to requirements engineering, requirement classification aims to categorize software requirements into different classes or types, such as functional and non-functional requirements. Dalpiaz et al. [19] constructed ML classifiers based on more general linguistic features (e.g., dependency types), and leveraged modern rule-based XAI tools to identify those features that appeared commonly and that helped distinguish functional and quality aspects.

[2]Due to the page limit, we only detail the most representative task in each phase. The complete version can be found in our supplementary materials online.

3.1.2 SE Tasks in Software Development. There are wide-ranging applications of XAI techniques in software development, encompassing tasks such as *Code Understanding*, *Program Synthesis*, and *Code Summarization*.

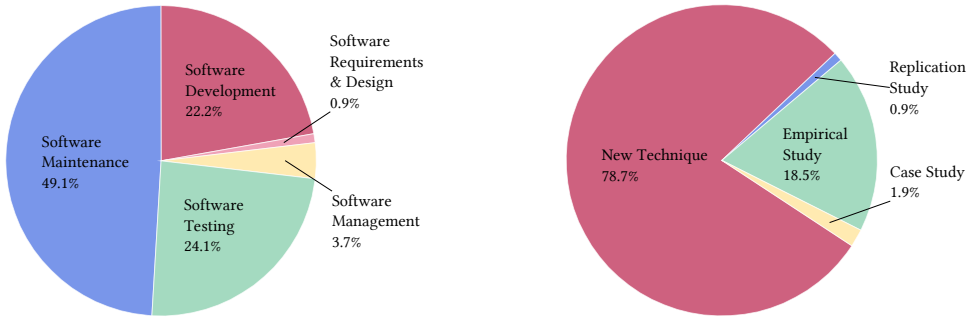
Code Understanding. Code understanding refers to the process of comprehending and analyzing source code deeply. Within the context of data-driven SE research, code understanding aims to seek an effective way to map source code into high-dimensional semantic space, thereby supporting a variety of code-centric downstream tasks. Inspired by the capability of complex AI models, deep neural networks in particular, in learning rich representations of raw data, a series of code models are trained on labeled (e.g., CodeSearchNet [49]) or unlabeled code corpus (e.g., CodeXGlue [76]). This training process produces code embeddings with rich contexts and semantics. Yang et al. [153] proposed **Graph Tensor Convolution Neural Network** (**GTCN**), a novel code representation learning model which is capable of comprehensively capturing the distance information of code sequences and structural code semantics, to generate accurate code embeddings. GTCN was self-explainable because the tensor-based model reduced model complexity, which was beneficial for capturing the data features from the simpler model space. Wan et al. [134] proposed three types of structural analysis, including attention analysis, probing on the word embedding, and syntax tree induction, to explore why the pre-trained language models work and what they indeed capture in SE tasks.

3.1.3 SE Tasks in Software Testing. Within the context of software testing, we found versatile applications of XAI techniques across a spectrum of tasks, including *Test Case-Related Automation*, *Debugging*, *Vulnerability Detection*, and *Bug/Fault localization*.

Vulnerability Detection. Software vulnerabilities, sometimes called security bugs, are weaknesses in an information system, security procedures, internal controls, or implementations that could be exploited by a threat actor for a variety of malicious ends. As such weaknesses are unavoidable during the design and implementation of the software, and detecting vulnerabilities in the early stages of the software life cycle is critically important. Benefiting from the great success of DL in code-centric SE tasks, an increasing number of learning-based vulnerability detection approaches have been proposed. To reveal the decision logic behind the binary detection results (vulnerable or not), most efforts focus on searching for important code tokens that positively contribute to the model's prediction. For example, Li et al. [65] leveraged GNNExplainer [160] to simplify the target instance to a minimal PDG sub-graph consisting of a set of crucial statements along with program dependencies while retaining the initial model prediction. Additionally, several approaches turn to providing explanatory descriptions to help security analysts understand the key aspects of vulnerabilities, including vulnerability types [34], root cause [123], similar vulnerability reports [93], and so on. Zhou et al. [175] proposed a novel contrastive learning framework based on a combination of unsupervised and supervised data augmentation strategy to train a function change encoder, and further fine-tuned three downstream tasks to identify not only silent vulnerability fixes, but also corresponding vulnerability types and exploitability rating.

3.1.4 SE Tasks in Software Maintenance. Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs. The applications of XAI in software maintenance are diverse, including *Malware/Anomaly Detection*, *Bug/Defect Prediction*, *Root Cause Analysis*, *Code Smell Detection*, *Bug Report-Related Automation*, and so on.

Bug/Defect Prediction. In the past few years, defect prediction is the most extensive and active research topic in software maintenance. According to different granularities, these studies can be further classified into two categories: file-level and commit-level (also known as **Just-In-Time (JIT)**) defect prediction. File-level defect prediction techniques often employ a set of hand-crafted



(a) Distribution of XAI4SE studies in different SE activities.

(b) Distribution of main contributions in different XAI4SE studies.

Fig. 2. Distribution of XAI4SE studies across different SE activities and contribution types.

feature metrics extracted from a software product to construct the classification model. For instance, Yang et al. [152] proposed a weighted association rule based on the contribution degree of features to optimize the process of rule generation, ranking, pruning, and prediction. By contrast, JIT defect prediction task aims to help developers prioritize their limited energy and resources on the most risky commits that are most likely to introduce defects. Zheng et al. [174] trained a random forest classifier based on six open-sourced projects as a JIT defect prediction model, and adopted LIME to identify crucial features. The evaluation experiments showed that the classifier trained on the five most important features of each project could achieve 96% of the original prediction accuracy.

3.1.5 SE Tasks in Software Management. There are four literature involving the utilization of XAI in software management, involving the following main SE tasks, i.e., *Mining Software Repositories*, *Configuration Extrapolation*, *Effort/Cost Estimation*, and *Developer Recommendation*.

Effort/Cost Estimation. Effort/Cost estimation predicts how much effort is required to complete a particular task or project. It is a crucial aspect of project management, playing a significant role in setting realistic timelines and allocating resources efficiently. A representative effort estimation activity is story point estimation, which is a regression task to measure the overall effort required to fully implement a product backlog item. Fu et al. [35] presented GPT2SP, a Transformer-based approach that captures the relationship among words while considering the context surrounding a given word and its position in the sequence. It is designed to be transferable to other projects while remaining explainable. They leveraged two concepts (i.e., feature-based explanations and example-based explanations) of XAI to (1) help practitioners better understand what are the most important word that contributed to the story point estimation of the given issue; and (2) search for the best supporting examples that had the same word and story point from the same project.

3.2 Exploratory Data Analysis

Figure 2(a) describes the distribution of 108 primary studies in five SE activities. It is noteworthy that the highest number of studies is observed in software maintenance, comprising 49.1% of the total research volume. Following that, 24.1% of studies were dedicated to software testing, and 22.2% of studies focused on solving SE tasks in software development. This distribution underscores the vital focus on development and maintenance tasks. By contrast, software requirements and design (0.9%) and software management (3.7%) only account for a marginal proportion of the research

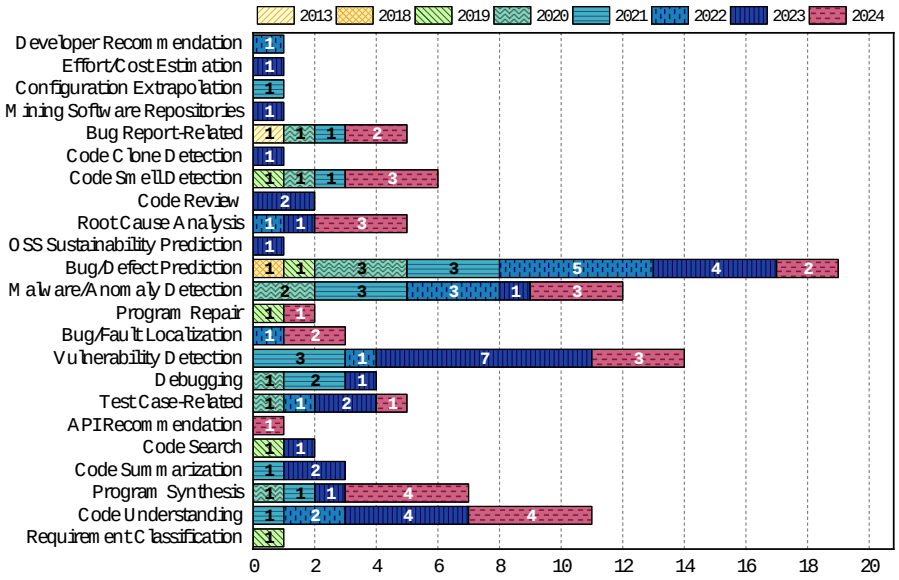


Fig. 3. Articles published per year according to SE tasks.

share, suggesting a relatively limited exploration in these areas. To further identify the main contribution of each primary study, we also investigated the contribution statements in each article, and then grouped them into four categories, i.e., *New Technique*, *Empirical Study*, *Case Study*, and *Replication Study*. As shown in Figure 2(b), 78.7% of the primary studies concentrated on proposing novel explanation techniques, while 18.5% of the research focused on leveraging off-the-shelf XAI tools to empirically study the explainability of certain AI4SE solutions from different perspectives (e.g., stability [117] and consistency [79]). Another two works [59, 109] performed case studies (1.9%) in real world, especially for enterprise usage, to investigate practitioners' adoption of AI4SE solutions. The remaining one [48] conducted a replication study (0.9%) to validate the controversial conclusions in terms of local and global explanations.

Figure 3 displays a visual breakdown of these SE tasks. Unsurprisingly, there was very little work done between before the years of 2012 and 2018. Early SE tasks to explore the explainability were those of *Bug Report-Related Automation* and *Bug/Defect Prediction*. It was not until 2020 that the diversity experienced a significant increase, including tasks such as *Malware/Anomaly Detection*, *Debugging*, and *Program Synthesis*. It is noteworthy that there are three main SE tasks that have consistently maintained high activity levels over the years: *Bug/Defect Prediction*, *Vulnerability Detection*, and *Malware/Anomaly Detection*, composing ≈42% of the studies we collected. We suspect that a variety of reasons contribute to the multiple applications of XAI in these tasks. First and foremost, is that these tasks are essentially binary classification problems that ML/DL models have shown promising results in. The second-largest reason is the mandatory requirement of high-stakes applications. In fact, black-box models are not even allowed in regulated fields unless they are supplemented with explanations [41]. In addition, several tasks (e.g., *Mining Software Repositories* [71], *Root Cause Analysis* [106, 125], and *Bug/Fault Localization* [56, 143]) that had yet to be explored or were underrepresented before [85] have recently gained traction of the research community, in part due to their relative success in practice. We anticipate that this trend continues as more powerful AI4SE solutions evolve from experimental prototypes to practical tools.

✍ ▸ RQ_1—Summary ◂

— We categorized a total of 108 primary studies into 23 unique SE tasks across five major activities within SDLC. Subsequently, we delved into the progress of existing XAI4SE research among these SE activities.
— Attention of academics and practitioners has experienced a notable shift across a 13-year period. Early studies predominantly concentrated on traditional classification tasks like *Bug/Defect Prediction*. Since 2021, the set of target SE topics grew to become more diverse and complex, including tasks such as *Developer Recommendation*, *Root Cause Analysis*, and *Program Synthesis*.
— While there has been a recent wealth of work, there are still underrepresented topics in software requirements and design and software management that should be considered by the SE community, suggesting a potential area of focus for future research in this field.

4 RQ_2: How XAI Techniques Are Used to Support SE Tasks?

In Section 3, we analyzed which AI-assisted SE tasks have been explored for explainability to date. In this part, we turn our attention to two key components of XAI: *explanation approaches* and *explanation formats*. Establishing the association between explanation approaches and target SE tasks helps to empirically determine whether certain XAI techniques are particularly suitable for specific SE tasks. Meanwhile, the explanation formats adopted across different SE tasks reveal key aspects that the stakeholders seek to understand from the decision of a given black-box model. Specifically, we aimed to create a taxonomy of XAI techniques for AI4SE studies and determine if there was a correlation between the explanation approaches and explanation formats.

4.1 RQ_{2a}: What Types of XAI Techniques Are Employed to Generate Explanations?

We first discuss various explanation approaches employed by existing XAI4SE studies. One classical practice is building a taxonomy of XAI techniques used in our surveyed literature. However, we note that the XAI community lacks a formal consensus on the taxonomy, as the landscape of explainability is too broad, involving substantial theories related to philosophy, social science, and cognitive science [119]. In addition, these taxonomies are mostly developed for general purpose or specific downstream applications such as healthcare [99] and finance [158], and may not be applicable to the SE field. As a countermeasure, we summarize the XAI techniques used in primary studies, and propose a novel taxonomy applicable to the field of SE. In particular, from an integration perspective, most XAI techniques studied in this review can be categorized into five groups: *OT* (≈34%), *IM* (≈23%), *DK* (≈20%), *AM* (≈10%), as well as a set of other custom, highly tailored approaches (≈13%). Figure 4 illustrates the various types of XAI techniques that we extracted from our selected studies.

OT. Embracing off-the-shelf techniques from the field of XAI served as a natural starting point for researchers, given the surge in publications and widespread adoption across various industries. Examining the prevalence of various different types of XAI tools, we found that **Feature Perturbation** [78, 108, 160] are the most popular approaches, followed by gradient-based [115, 126] and decomposition-based approaches [5, 86, 118]. The prevalence of perturbation-based approaches is expected, as they can work at various levels including embeddings vectors [149], source code [140], texts [114], and data structure [65], which are common types of artifacts being used in AI4SE approaches. An early representative perturbation-based approach is **Local Interpretable Model-agnostic Explanations (LIME)** [108]. Specifically, LIME first perturbs the to-be-explained instance in the high-dimensional feature space to randomly generate synthetic neighbors. Then, based on their prediction results derived from the global black-box model, LIME trains a local

Fig. 4. XAI technique & distribution.

surrogate model (e.g., decision tree and linear regression) to produce an explanation. LIME has been successfully applied to various SE tasks, such as defect prediction [61, 140, 174], OSS Sustainability Prediction [149], and test case generation [2]. **SHapley Additive exPlanations (SHAP)** [78], which stems from game theory, is another popular XAI technique based on perturbation-based feature attribution. It assigns each feature a fair value, otherwise known as *shapley value*, to measure its contribution to the model's output. Features with positive SHAP values positively impact the prediction, and vice versa. Similar to LIME, SHAP is also model-agnostic, thus it can be used to explain any ML model. For example, Widyasari et al. [144] applied a tree ensemble model-specific variant, TreeSHAP [77], to identify which code statements are important in each failed test case. However, post-hoc approaches such as LIME and SHAP are computationally expensive. As a consequent, they are usually limited to simpler problems with a small number of features. In addition, important features highlighted by out-of-the-box tools are not necessarily user-friendly in terms of understandability and usability. For example, the importance of a single token in a code snippet may not convey a sufficiently meaningful explanation.

IM. As pointed out by Chen et al. [11], complex models did not always perform better than simpler alternatives. Thus, for certain SE tasks, the easiest way to achieve explainability is to construct IMs, such as **Decision Tree** (**DT**), **Linear Regression** (**LR**), and **Naïve Bayes** (**NB**). These models have built-in explainability by nature. For instance, DT predicts the value of a target variable by learning simple `if-then-else` rules inferred from the data features. The tree structure is ideal for capturing interactions between features in the data, and also has a natural visualization of a decision making process. Taking Figure 5 as an example, each node in a DT may refer to an explanation, e.g., when the `time_days` variable (i.e., the number of days to fix the bug) is greater than 13.9 and the last status is `Resolved Fixed`, then the bug will be re-opened [116]. In addition, IMs can serve as post-hoc surrogates to explain individual predictions of black-box models. The goal behind this insight is to leverage a relatively simpler and transparent model to approximate the

predictions of the complicated model as best as possible, and at the same time, provide explainability. Surrogate models have shown effectiveness in explaining AI4SE approaches built upon more complex ML/DL models such as deep neural networks. Examples include vulnerability detection [177], defect prediction [103], and program repair [83]. It is noteworthy that, due to the weak capability of processing complex data, intrinsically IMs are prone to getting trapped into the tradeoff dilemma between performance and explainability, i.e., sacrificing predictive accuracy in exchange for explainability.

DK. Inspired by the successful practice in **Software Engineering for AI (SE4AI)**, some works take special steps to incorporate additional knowledge from experts into their explanations. Concretely, we identify two incipient trends in the application of DK: (❶) designing one or more auxiliary tasks related to the main task to provide additional insights regarding the input data, and (❷) the use of an external knowledge database curated by experts. For example, to explain why a program/commit was predicted as vulnerable, recent works proposed to predict vulnerability types [34], identify key aspects [123], generate vulnerability descriptions [82, 166], search for similar issues [93], and so on. To assist developers in understanding the return results of neural code search tools, XCoS [135] constructed a background knowledge graph, and regarded it as an external knowledge base to provide conceptual association paths, relevant descriptions, and additional suggestions, as explanations. Despite promising, they are mostly task-specific, and "*what makes a good explanation*" is still an open problem.

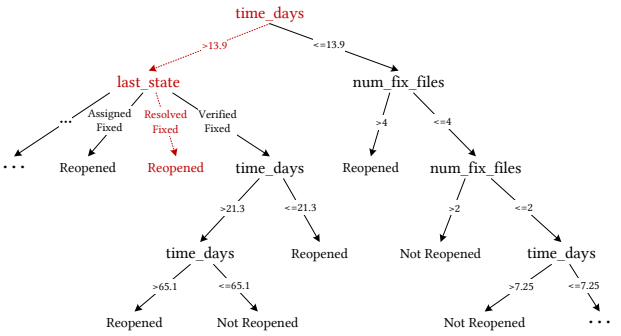


Fig. 5. Sample decision tree used for re-opened bug prediction.

AM. As an increasingly common ingredient of neural architectures, AM has been widely applied to various SE tasks. Besides providing substantial performance benefits, it allows users to understand which parts of an input a model is most interested in through assigned weights, making the use of attention an intuitive option in practice. For example, Li et al. [63] employed an attention-based GNN model, named GAT [132], to weigh the importance of neighboring code graph nodes in runtime exception detection. Apart from explaining a model's individual predictions, AM can also offer insights into the inner workings of foundation models, providing a powerful tool for SE in the era of **Large Language Models (LLMs)**. A representative technique is probing, which trains a shallow classifier on top of the pre-trained or fine-tuned LLMs to identify certain knowledge/linguistic properties acquired by the model. For instance, Ma et al. [81] designed four probing tasks to analyze the capabilities of code models in understanding syntax and semantics by directly recovering the syntax and semantic structures from the code representation.

Although attention is a core component of Transformers and has been integrated into various neural architectures such as RNNs and GNNs, its usefulness for explainability remains a topic of ongoing debate. There is skepticism about whether the AMs, which require substantial retraining, evaluation, and validation, can significantly enhance explainability in XAI4SE. Recent studies [58, 98] have highlighted a persistent misalignment between human attention and model-generated attention in all CodeLMs, unveiling the faithfulness violation issue. Moreover, raw attention contains redundant information, further reducing its reliability in explanations [171].

Others. Apart from the above approaches, we also observed a number of XAI techniques, such as *Causal Inference* and *LLM-based Chatbots*, that are specifically tailored for given SE tasks or neural architectures. Theory of Causation [131] endows the model with the ability to pursue real causality without the interference from confounding factors. Such mechanism is a useful verification tool to achieve a more complete understanding of black-box neural models in SE tasks. For example, Palacio et al. [90] proposed a post-hoc approach specific to neural code models that provides programming language-oriented explanations based upon causal inference. To further provide actionable advice, some works employed counterfactuals-based causal inference paradigm to understand how the model reacts to feature changes. Cito et al. [17] proposed a **Masked Language Modeling** (**MLM**)-based perturbation algorithm, which replaces each token with a blank "mask" and uses MLM to come up with a plausible replacement for the original token, for generating counterfactual explanations. Counterfactuals-based explanation not only reveals which region of the input program is used by the code model for prediction, but also conveys critical information on how to modify the code so that the model will change its prediction. Recently, inspired by the remarkable performance in natural language understanding and logic reasoning, LLMs have been (in)directly integrated into the workflow of AI4SE approaches to offer explainability [74, 143]. For instance, given an identified TODO-missed commit, Wang et al. [136] fed it with the most similar historical commit to ChatGPT[3] to analyze *where* and *what* comment users should insert.

4.1.1 Exploratory Data Analysis. Overall, *OT* is the most prevalent explainability technique in XAI4SE research, followed by *IM*, *DK*, and *AM*. Given the rapid development of the XAI community, the popularity of OT and AM is not surprising. Meanwhile, the prevalence of IM is also logical, as they are inherently more understandable to humans and can serve as surrogate models to explain individual predictions of complex deep neural networks, which are more common in AI4SE approaches. Due to the diversity of SE data, DK-based explanations are also popular. The explanations are intended to give control back to the user by helping them understand the model and offering additional insights into the input data.

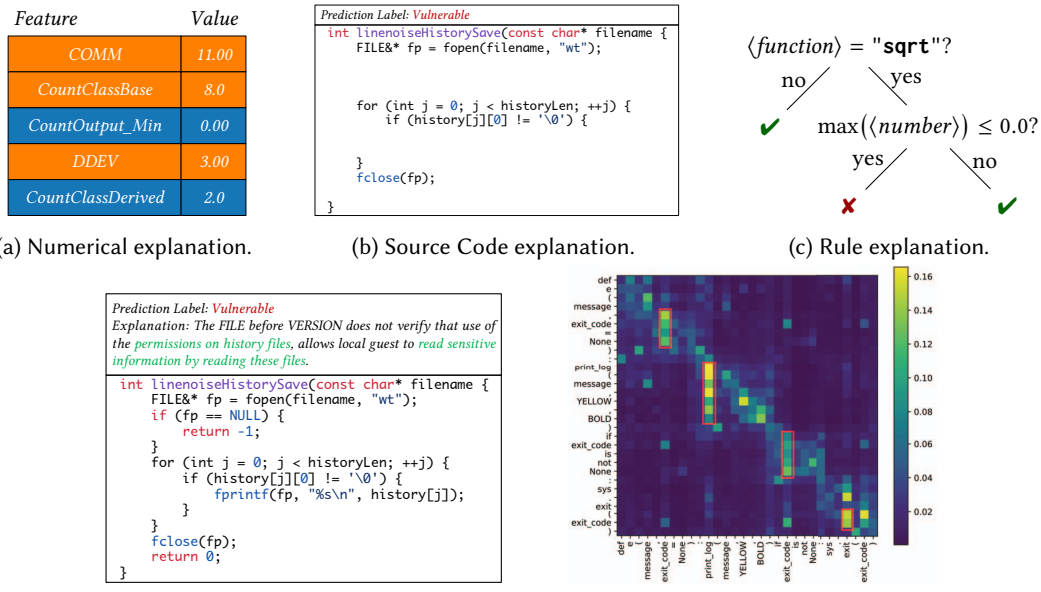
In addition to the prevalence, selecting the most suitable explanation approach for a specific task is also a crucial aspect that needs to be carefully considered. We further extracted the selection rationale for XAI techniques from the primary studies, and classified them into the following three categories:

Task Fitness. Given the inherent differences in feature engineering and functional requirements among various AI4SE workflows, some studies selected XAI techniques based on their characteristics and fitness with target SE tasks. For instance, the explanations for most of the feature engineering-based AI4SE pipelines (e.g., defect prediction [103] and OSS sustainability prediction [149]) were derived by using IMs, such as decision tree or RuleFit [33], thanks to their strong ability to analyze and extract human-understandable rules from hand-crafted feature metrics which are usually limited.

Model Compatibility. Since certain crafted XAI techniques have strict application scenarios, e.g., requiring the internal architecture or parameter information of to-be-explained models, some researchers determined the most suitable XAI techniques from those compatible with their employed models. For instance, deconvolution and Grad-CAM are preferred in CNN-based SE models [44, 107], while CodeQ [96] is less commonly used in resource-constrained scenarios due to its expensive computational overhead.

Stakeholder Preference. In addition to the task fitness and model compatibility, stakeholder preference is also one of the important factors affecting the selection of XAI techniques. Generally

[3] https://chatgpt.com/



(a) Numerical explanation. (b) Source Code explanation. (c) Rule explanation.

(d) Textual explanation. (e) Visual explanation.

Fig. 6. Different formats of explanation.

speaking, model users aim to utilize XAI techniques to better understand the output of a deployed model and make an informed decision, while designers focus on using XAI techniques during model training and validation to verify that the model works as intended. In addition, the explanations would be generated for distinct purposes at different levels of expertise even when considering a single stakeholder. For instance, to assist software developers in understanding a defective commit, some approaches simply highlight the lines of code that the model thinks are defective [140], while others extract human-understandable rules [103], or even natural language descriptions [82] from the defective code that can serve as actionable and reusable patterns or knowledge.

4.2 RQ$_{2b}$: What Format of Explanation Is Provided for Various SE Tasks?

Next, to analyze the formats of explanation being used in XAI4SE research, we provide a high-level classification, along with descriptive statistics, as to why some formats of explanation were used for particular SE tasks. In total, we observed five major explanation formats: *Numeric*, *Text*, *Visualization*, *Source Code*, and *Rule* as illustrated in Figure 6.

Numeric. Numerical explanations, which are capable of conveying information in a compact format, focus on quantifying the positive or negative contribution of an input variable to the model prediction. Such importance scores can either be directly used as explanations [53, 174] or serve as indicators to guide key feature selection [124]. Popular examples of numerical explanations are LIME [108] and SHAP [78]. Esteves et al. [26] used SHAP values to understand the CK metrics [14] that influenced the defectiveness of the classes. In the same context, Lee et al. [61] employed three widely-used model-agnostic techniques, including LIME, SHAP, and BreakDown [122], to calculate the contribution of each feature for defect models' predictions. Xiao et al. [149] leveraged the local explanations generated by LIME from the XGBoost model to analyze the contribution of variables to sustained activity in different project contexts. To reflect the global behavior of a complex JIT defect prediction model, Zheng et al. [174] employed SP-LIME, a variant of LIME, to analyze the

relationship between the features in the model and the final prediction results. SP-LIME explicitly selects representative and diverse (but not repetitive) samples, presenting a global view within the allocated budget of maximal features. Sun et al. [124] used SHAP to guide the search for the feature that has largest malicious magnitude, i.e., having the potential to be manipulated by the adversary, to test the robustness of malware detectors.

Text. In contrast to numerical explanations, textual natural language descriptions are easier to be comprehended by non-experts, offering clarity in understanding the behavior of intelligent SE models. Such textual explanations can either be derived from scratch by using generative models [82, 123, 166] or retrieved from external knowledge bases [71, 147]. For example, Zhang et al. [166] leveraged a GRU-based decoder to generate vulnerability symptom- or reason-related descriptive sentences step by step. Such summarization-styled explanations can effectively bridge the cognitive gap between structured programming language and flatten natural language. Wu et al. [147] built a semantic database based on malware key features and functional descriptions in developer documentation, and leveraged the mapping relation between the malicious behaviors and their corresponding semantics to generate reasonable descriptions that are easier for users to understand. Inspired by similar/homogeneous vulnerabilities that have similar root causes or lead to similar impacts, Ni et al. [93] first retrieved the most semantically similar problematic posts from SO and prioritized the most useful response based on a quality-first sorting strategy. Then, they employed the BERT-QA model [22] to extract the root cause, impact, and solution from the answers to the given questions as useful and understandable natural language explanations. Compared to generative models pre-trained on the human-labeled corpus, external knowledge bases such as Stack Overflow and Wikipedia[4] offer structured knowledge models that explicitly store rich factual knowledge. Thus, they are well-known for their symbolic reasoning ability, which generates explainable results, and avoids hallucinations originating from generated statements that are factually incorrect.

Visualization. Besides explaining through numerical importance scores and textual natural language descriptions, users can understand the behavior of the underlying model through the form of visuals. Humans, in general, can process visual information faster and much easier as compared to other information [88]. Common techniques involve visualizing attention heads for a single input using bipartite graphs or heatmaps. They are simply disparate visual representation of attentions, one as a graph and the other as a matrix. Wang et al. [138] visualized the attention weights of the most important words and phrases that have contributed to the model's predictions. In addition, some approaches developed interactive **User Interface (UI)** to provide visual explanations [51, 135]. For instance, Jiang et al. [51] designed several contribution mining algorithms to infer the key elements in code that contribute to the generation of the key phrases in the comments. When a developer intends to comprehend the code, the UI loads the auto-generated comments and presents to the developers the graphic illustration by coloring the important phrases and the corresponding parts in the source code. In this way, the developer can check whether the auto-comments correctly describe the intention of the code.

Source Code. Some attempts borrowed certain classical techniques, such as program mutation [121] and **Delta Debugging (DD)** [163], from the field of software testing to search for important code snippets positively contributing to the model predictions. For instance, Geng et al. [38] identified important code tokens contributing to the generation of a specific part of the summarization by checking which meaningful words disappeared in any of the summarization newly generated from mutants. Rabin et al. [104] leveraged DD to simplify a piece of code into the minimal fragment

[4] https://www.wikipedia.org/

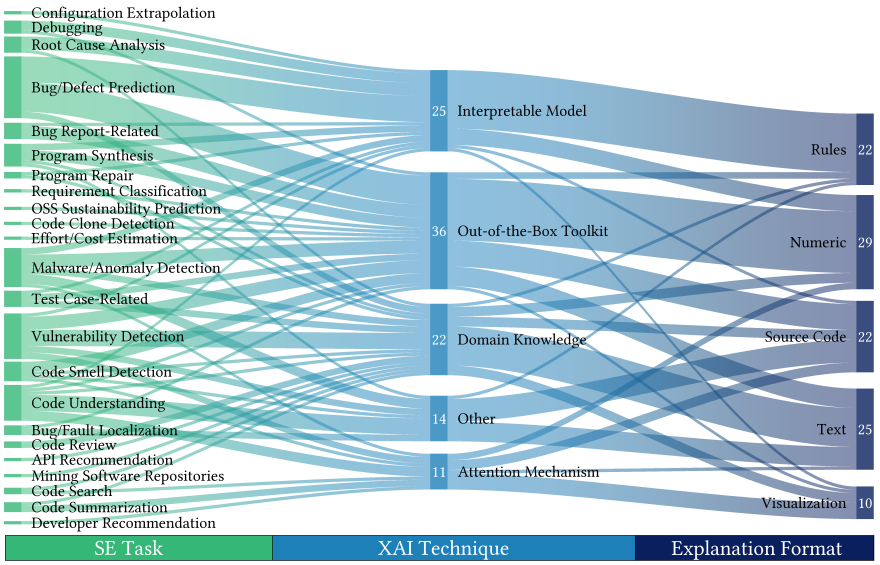


Fig. 7. Explanation formats by SE task and XAI techniques taken.

without reversing it original prediction label. The reduction process continues until the input data is either fully reduced (to its minimal components, depending on the task) or any further reduction would corrupt the prediction. In contrast to debugging-based techniques that can be applied to any DL architecture, Li et al. [65] employed a GNN-specific explanation framework, GNNExplainer [160], to simplify the target code instance to a minimal statements subset.

Rule. Rule, which can be organized in the form of IF-THEN-ELSE statements with AND/OR operators, is a schematic and logic format. Despite its complexity compared to visualization and natural language description, rule-based explanation is still intuitive for humans and useful for expressing combinations of input features and their activation values. Generally, these rules approximate a black-box model but have higher interpretability. Zou et al. [177] identified important code tokens whose perturbations lead to the variant examples having a significant impact on the prediction of the target model via heuristic searching, and trained a decision tree-based regression model to extract human-understandable rules for explaining why a particular example is predicted into a particular label. To explain the individual predictions of the black-box global model, Pornprasit et al. [103] built a RuleFit-based local surrogate model, which combined the strengths of decision tree and linear model, to understand the logical reasons learned from the rule features. Cito et al. [16] proposed a rule induction technique which produced decision lists based on features and mispredicted instances to explain the reasons for mispredictions.

4.2.1 Exploratory Data Analysis. Figure 7 shows a breakdown of the relationships between the SE tasks, the explanation techniques taken, and explanation formats. We found that the distribution of different explanation formats is relatively even except *Visualization* (≈9%), and the most common format being used is *Numeric* (≈27%). The primary reason is that numerical features are a common source of information across all aspects of data-driven methodologies. SE tasks such as *Bug/Defect Prediction* and *Code Smell Detection* commonly use a collection of hand-crafted numerical features (e.g., code metrics [14], smells [100] and permission and API calls [101]) to train a model. Measuring the relevance of each input feature to a model's prediction is intuitive and has been well established within the field of XAI, hence it is not surprising that the majority of reviewed studies focus on

this format. We also noticed that visual explanation is less commonly used, only accounting for a total of 10 primary studies. One possible reason contributing to its relative lack of adoption lies in that, although visualization can provide a fast and straightforward explanation for practitioners (e.g., a developer, domain expert, or end-user) who are inexperienced in ML/DL [120], it can only convey limited information and requires post-processing for further use.

In addition to the prevalence, we observed that the formats of explanation varied even in a single SE task. As an example, the explanations generated for binary vulnerability detectors in our surveyed studies can be text (e.g., vulnerability descriptions [93, 166] and types [123, 175]), source code (e.g., code statements [65, 127]), or rules [177]. This diversity helps to satisfy the personalized needs of the stakeholders who have different intents and expertise.

✍ ▸ RQ_2—Summary ◂

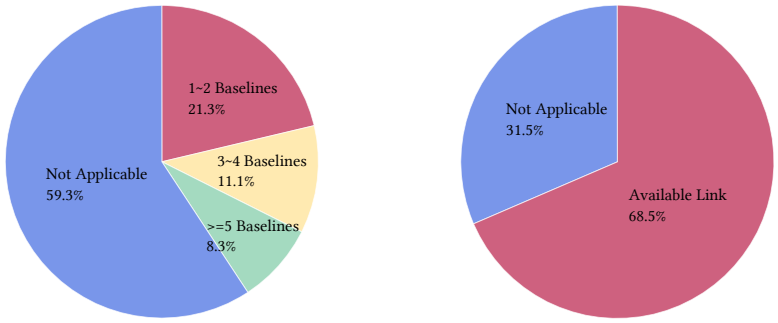
- Our exploratory data analysis revealed five commonly used XAI techniques, including *OT* (≈34%), *IM* (≈23%), *DK* (≈20%), *AM* (≈10%), as well as a subset of other custom, highly tailored approaches (≈13%).
- We summarized the selection strategies for XAI techniques in SE tasks into three main categories: *Task Fitness*, *Model Compatibility*, and *Stakeholder Preference*.
- A variety of explanation formats have been explored in our surveyed studies, with the main formats utilized being *numeric* (≈27%), *text* (≈23%), *visualization* (≈9%), *source code* (≈20%), and *rule* (≈20%).
- We found a strong correlation between the SE tasks, the explanation techniques taken, and explanation formats. Additionally, the formats of explanation also varied even in a single SE task. This diversity helps to satisfy the personalized needs of the stakeholders who have different intents and expertise.

5 RQ_3: How Well Do XAI Techniques Perform in Supporting Various SE Tasks?

Evaluating the effectiveness of proposed solutions against existing datasets and employing baseline comparisons is a standard practice in AI4SE research. In this RQ, we endeavor to investigate the influence of XAI4SE research by scrutinizing the effectiveness of techniques proposed in the studies under consideration. Our analysis primarily focuses on evaluating metrics on a task-specific basis, aiming to encapsulate the prevailing benchmarks and baselines within the field of XAI4SE research.

5.1 RQ_{3a}: What Baseline Techniques Are Used to Evaluate XAI4SE Approaches?

For RQ_{3a}, we scrutinize the baseline techniques employed for comparative analysis in our selected primary studies. Although common baselines for particular SE tasks were identified, it was noted that a substantial portion of the literature autonomously developed unique baselines. The extensive variety and volume of these baselines precluded their detailed inclusion within the body of this manuscript. Therefore, we included the listing of baselines that each article compared against on our interactive website at https://riss-vul.github.io/xai4se-paper/. Figure 8(a) briefly analyzes the distribution of baseline usage in XAI4SE studies. Approximately ≈60% of the studies reviewed do not engage in comparisons with any baseline, whereas a minority contrasts their findings with more than four distinct methods (≈8.3%). It was observed that numerous baseline techniques consist of established white-box models with transparent algorithms or conventional expert systems. This trend may be attributed, in part, to the nascent stage of XAI4SE research, which has resulted in a limited range of existing XAI-centric comparatives. As XAI4SE progresses towards maturity, an evolution towards evaluations incorporating benchmarks against established XAI-centric methodologies is anticipated. Furthermore, it was noted that the selection of baseline techniques exhibits a



(a) Distribution of baseline usage. (b) Approach availability.

Fig. 8. Statistical information of baseline techniques and XAI4SE approaches.

high degree of specificity, varying significantly even among studies addressing identical SE tasks. For example, to evaluate the effectiveness of their proposed explainable vulnerability detection approaches, eight out of 14 primary studies employed at least two baselines for evaluation, while only two articles [15, 46] overlap slightly in terms of the baselines.

A concerning trend identified in our review is the lack of publicly accessible implementations for many XAI4SE approaches. We conducted a manual inspection of all links provided within each study. In instances where a link to a replication package was available, we assessed its contents for source code and relevant documentation. Absent any direct links, we also endeavored to locate either the original replication package or an equivalent reproduction package on GitHub using the title of the article as a search query. As depicted in the pie chart in Figure 8(b), approximately only 68% of the primary studies offer accessible replication packages. Among the remaining studies with replication packages, a considerable portion of them propose XAI techniques for specific SE tasks for the first time, such as *Mining Software Repositories* [71] and *OSS Sustainability Prediction* [149]. This, in part, explains the proliferation of highly individualized baseline approaches. Researchers often lack access to common baselines for comparison, compelling them to implement their own versions. The robustness of results in such articles may be compromised, as many do not provide information about the baselines used. Moreover, distinct implementations of the same baselines could lead to confounding results when assessing purported improvements. While we anticipate that the set of existing publicly available baselines will improve over time, we also recognize the necessity for well-documented and publicly available baselines, accompanied by guidelines that dictate their proper dissemination.

5.2 RQ$_{3b}$: What Benchmarks Are Used for These Comparisons?

For RQ$_{3b}$, we investigated the collection strategies of benchmarks in XAI4SE studies. As can be seen from the data in Figure 9 (left), only 36 (≈33%) studies used previously curated benchmarks for evaluating XAI4SE approaches. The selection of open-source benchmarks is often motivated by their compelling nature in assessing the performance of AI-driven methodologies, which facilitates the reproducibility and replication by subsequent studies. Given the nascent emergence of XAI4SE research, there is an observed scarcity of appropriate benchmark datasets. This also explains why there is a considerable amount (28.7%) of self-generated benchmarks. This trend within XAI4SE is worrying, as there are few instances where XAI approaches can appropriately compare against one another with available benchmarks. In our online repository, we recorded the accessible benchmark links provided by primary studies. Our aim is to assist researchers by offering insights into the available benchmarks for evaluating methodologies within distinct SE tasks. Furthermore, we

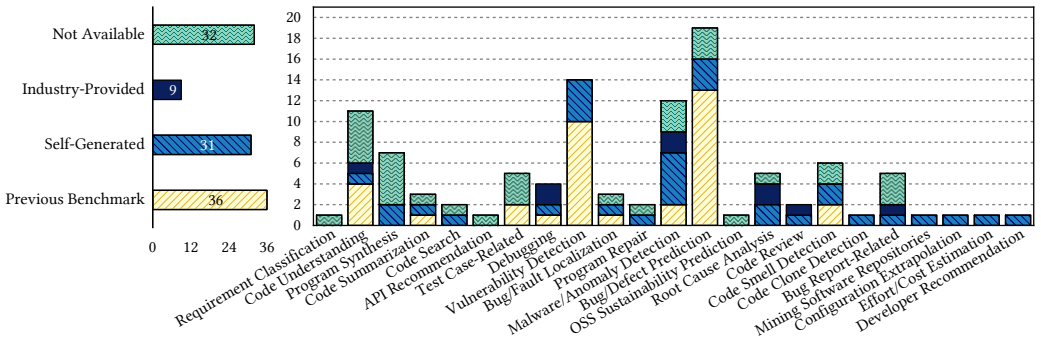


Fig. 9. Collection strategies of benchmarks by SE task.

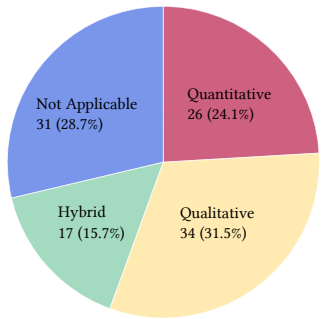


Fig. 10. Evaluation strategies of XAI techniques.

advocate for future scholars to share their self-created benchmarks publicly, thereby furnishing a valuable resource that facilitates not only comparative analyses among different methodologies but also broadens the dataset accessible for Explainable AI techniques.

While the adoption of pre-existing benchmarks was infrequent across our surveyed studies, we did observe a subset of benchmarks that recurred within our primary studies. A comprehensive delineation of the types of benchmarks employed in these primary studies is depicted in Figure 9 (right). For vulnerability detection, we found that the Big-Vul [31] dataset was used frequently, including evaluating the accuracy of the key aspects extracted from the detected vulnerabilities (e.g., vulnerable statements [46, 65] and vulnerability types [34]). Additionally, the dataset released by Yatish et al. [157] was employed for benchmarking prior XAI techniques targeting defect prediction models [53, 61].

5.3 RQ_{3c}: What Evaluation Metrics Are Employed to Measure XAI4SE Approaches?

With an increasing number of XAI techniques, the demand grows for suitable evaluation metrics [3, 7, 42]. This need is not only recognized by the AI community, but also by the SE community as evaluating the performance of XAI techniques is a crucial aspect of their development and deployment [109]. Figure 10 describes the distribution of evaluation strategies found in this SLR. The absence of objective, quantifiable evaluation is not surprising, given the scarcity of reliable benchmarks that are publicly available, as discussed above. In our analysis of utilized evaluation strategies within work on XAI4SE, we observed that nearly one-third (≈32%) of primary studies only adopted anecdotal evidence or user study, instead of quantitative metrics to evaluate their proposed approaches. That's to be expected because traditional performance metrics exist to evaluate

Table 3. Metrics Used for Evaluation

Property	Description	Metric	References
Correctness	How faithful the explanation is w.r.t the black box.	%Consistency	[103, 151]
		PCR	[12, 177]
		Statistical Test	[103, 151, 162]
		Distance-based	[103, 151, 162]
		R^2	[53, 69]
		RBO	[61, 162]
Consistency	How deterministic the explanation approach is.	Statistical Test	[4, 69, 79]
Continuity	How continuous and generalizable the explanation function is.	Similarity-based	[46, 66]
Contrastivity	How discriminative the explanation is w.r.t. other events or targets.	%Unique	[103, 151]
Compactness	The size of the explanation.	Reduction Ratio	[104, 139]
		#Rules	[55, 83, 102]
Coherence	How accordant the explanation is with prior knowledge and beliefs.	Alignment-based	[58, 82, 123, 143, 166]
		Classification-based	[9, 15, 16, 34, 39, 46, 113, 175]
		Ranking-based	[65, 140, 140, 170]
		Statistical Test	[1, 51, 97, 98]
Efficiency	The average runtime of generating one explanation per instance.	Runtime	[12, 104, 139, 162]

prediction accuracy and computational complexity, while auxiliary criteria such as explainability may not be easily quantified [27]. Among 43 articles that performed quantitative evaluation, we found that the SE community also has yet to agree upon standardized evaluation metrics due to the absence of ground truths. Furthermore, due to the wide range of objectives associated with explainability in SE tasks, relying solely on a single evaluation metric may not adequately reflect the full spectrum of an XAI tool's performance. Consequently, researchers frequently utilize a variety of evaluation metrics, each designed to measure specific aspects of explainability. According to the best practice within the field of XAI [91], these quantitative metrics can be clustered from a multi-dimensional view, named **Co-12 properties**. Table 3 describes each Co-12 property we identified, while listing the evaluation metrics that were mainly related with this property and the articles that applied these metrics. Since over 30 unique metrics are adopted by our collected primary studies, which is difficult to present them all in this SLR, we listed 16 main metrics across six major Co-12 properties, where each metric was researched by no less than two studies.

Correctness. Correctness describes the degree at which an explanation technique approximates the behavior (either locally or globally) of the target model. In certain primary studies, it is also referred to as **Fidelity** [13, 66]. Common metrics chosen to evaluate correctness include *%Consistency*, ***Positive Classification Rate (PCR)*** [43], *Statistical Test*, *Distance-based*, R^2, and ***Ranked Biased Overlap (RBO)*** [141]. For instance, RBO is a similarity measure (assessed within the ranges of [0, 1]) that quantifies the differences between indefinite ranked lists. A higher value signifies stronger alignment or closeness between the explanations generated by different explainers. Given that numerical explanation (e.g., feature importance) is one of the most common explanation formats in our surveyed articles, the quantification of differences between feature importance is crucial for assessing model trustworthiness.

Consistency. In addition to correct results, the generated explanations need to be consistent. That is, two models that give the same outputs for all inputs should have the same explanations in order to be useful for an expert. A common evaluation approach is statistical test, which computes the statistical differences among multiple runs on the same instance. For example, Awal et al. [4] assessed inconsistency by running the same explanation technique on 500 instance (100 consecutive iterations per instance) and measuring the rank difference of each metric. To measure whether the explanation results between two executions were statistically different, they applied the Wilcoxon Signed-Rank Test [145] and utilized Cliff's $|\delta|$ effect size [18] to quantify the extent of the differences.

Continuity. Previous work [40] has demonstrated that slight variations in the input samples can confuse the explanation results. If the model response to similar samples varies significantly, not only will it not convince the experts, but it will lead them to doubt the reliability of the underlying predictions. Consequently, Continuity is proposed to describe how continuous and generalizable the explanation function is. In our identified studies, similarity-based metrics, such as *Dice Coefficient* [46] and *Jaccard Similarity* [66], are often utilized to measure the continuity of generated explanations. Higher continuity indicates a better generalizability to new contexts.

Contrastivety. Contrastivety aims to describe the discriminativeness of an explanation. Intuitively, the explanation for a particular target or model output should be different from an explanation for another target. *%Unique* is the most frequent metric, used in two studies [103, 151]. %Unique measures the percentage of unique explanations generated by each technique. The higher percentage of unique explanations indicates that the explainer can effectively generate a more specific (i.e., less duplicate) explanation to the target instance.

Compactness. To avoid increasing the human cognitive load, explanations should be sparse, short and not redundant. As a consequence, compactness is defined to measure the size of generated explanations. Metrics like *Reduction Ratio* and *#Rules* are the most commonly used, appearing in two and three studies, respectively. For instance, Rabin et al. [104] employed size reduction ratio of input programs to evaluate the conciseness of generated explanations.

Coherence. In many scenarios, even when interacting with the same model, stakeholders who have different intents and expertise may consume explanations for distinct objectives. Hence, assessing to what extent the explanation is consistent with relevant background knowledge, beliefs and general consensus is necessary. The most commonly used coherence metrics are *Alignment-based* (e.g., ROUGE-L and BLEU), *Classification-based* (e.g., Accuracy, Precision, and Recall), *Ranking-based* (e.g., Top K-based and MRR), and *Statistical Test*. For example, Fu et al. [34] employed Accuracy and Weighted F1-score to evaluate the performance of vulnerability classification. Sun et al. [123] adopted ROUGE-1/2/L to measure the usefulness of generated key aspects for alert prediction.

In addition to these popular functional metrics, we also observed a limited number of non-functional metrics. One representative example is **Efficiency**, which is used in nine studies [12, 104, 139, 162]. Most SE tasks, such as code search and code completion, have a high demand for the response efficiency of AI models, i.e., preferring tools/approaches providing actionable results within acceptable time cost, and explanations are no exception. Given that explainability mostly serves as a by-product of model outputs, approaches requiring higher computation overhead are unlikely to be adopted by the audience, even if they achieve promising performance in terms of other aspects.

✍ ▸ RQ_3—Summary ◂

— The analysis indicated a notable scarcity of well-documented and reusable baselines or benchmarks for work on XAI4SE. Approximately 28.7% of the benchmarks employed in the evaluations of our studied approaches were self-generated, with a significant portion not being publicly accessible or reusable.
— We noticed that there is no consensus on evaluation strategies for XAI4SE studies, and in many cases, the evaluation is only based on specific properties, such as correctness and coherence, or researchers' subjective intuition of what constitutes a good explanation.

6 Discussion

In this section, we discuss current challenges (Section 6.1) and highlight promising opportunities (Section 6.2) for conducting future work on XAI4SE.

6.1 Challenges

Challenge 1: Lack of Consensus on Explainability in SE. One of the major challenges in developing explainable approaches for AI4SE models is the lack of formal consensus on explainability within the field of SE. As shown in the earlier sections, numerous points of view are proposed when trying to articulate explainability for a specific SE task. For instance, to assist software developers in understanding defective commit, some approaches simply highlight the lines of code that the model thinks are defective [140], while others extract human-understandable rules [103], or even natural language descriptions [82] from the defective code that can serve as actionable and reusable patterns or knowledge. Although this diversity can meet the distinct requirements of audiences with different levels of expertise, it greatly increases the difficulty of establishing a unified framework which provides common ground for researchers to contribute toward the properly defined needs and challenges of the field.

Challenge 2: Tradeoff between Performance and Explainability. For some real-world tasks, a model with higher accuracy usually offers less explainability. Such ***Performance-Explainability Tradeoff (PET)*** dilemma often results in user hesitation when choosing between black-box models and inherently transparent models. While certain studies [110, 111] indicate that black-box models performing complex operations do not necessarily result in better performance than simpler ones, it is often the case for advanced SE models built upon immense amounts of structured and unstructured data. This challenge highlights the necessity of flexibly selecting XAI techniques according to various factors, such as the characteristics (e.g., input/output format and model architecture) of different SE tasks, resource availability (e.g., time cost), and consideration of risk (e.g., ethics and legality). For example, explainability may be particularly important to avoid bias in a developer recommendation system, while performance might be prioritized in an AI-aided coding scenario.

Challenge 3: Disconnection between Academic Efforts and Industry Needs. The deployment of XAI techniques in SE practice, particularly security-critical tasks like vulnerability detection [10], necessitates rigorous validation to satisfy not only effectiveness, but also robustness, controllability, and other special concerns. Here, the controllability means that each user should be shown the most complex explanation this user can still grasp, i.e., giving control back to the user. This property is important because users can adapt the explanation to their needs [32]. This proves challenging given the complex and dynamic nature of SDLC. In addition, due to the different intents and expertise of audiences, a single explanation format may not be applicable to everyone. For instance, while visual explanation can be attractive for the layperson since it is natural and intuitive, it could potentially overwhelm domain experts with superfluous information, thereby increasing their cognitive load and rendering the tool counterproductive [158].

6.2 Opportunities

Opportunity 1: Application on Underexplored and Complex SE Tasks. Throughout our study, it was clear that XAI techniques have been widely used in certain SE tasks to support users in decision-making or improve the transparency of AI models. However, the current application of XAI techniques in some SE activities remains relatively sparse. As shown in Figure 2(a), not many studies focus on software requirements and design and software management. This unveils a substantial opportunity: broadening the application of XAI techniques to these under-explored

research topics. We suggest that future work should concentrate on two aspects. First, for emerging SE tasks that have only recently benefited from ML/DL, integrating off-the-shelf XAI techniques into existing research workflows as a component rather than developing end-to-end solutions appears more pragmatic. For example, compared to conventional tools which heavily rely on pre-defined templates or grammars, leveraging the powerful capability of LLMs on code and natural language comprehension to generate formal program specifications [80], fix vulnerabilities [148], and analyze developers' sentiment [169] have shown promising results. By combining them with established techniques [84], we can achieve reliable and efficient explanations in a seamless manner. Second, for complex SE tasks that have yet to be fully explored by the research community, such as *Code Search* and *API Recommendation*, it is promising to conduct user survey (e.g., interviewing relevant industrial practitioners) to understand their perceptions.

Opportunity 2: Customizing Task-oriented XAI. In the midst of our analysis it became clear that most approaches used to provide explanations for AI-driven SE models are directly inherited from off-the-shelf XAI techniques without any customization. Unfortunately, existing XAI techniques, originally not designed for SE tasks, likely generate suboptimal or even misleading explanations. Given these reasons, we recognize a research opportunity to customize explanation approaches more suitable for SE tasks. In this regard, the integration with DK appears to be the most promising direction to explore. For example, security experts can construct a vulnerability knowledge base by extracting multi-dimensional information from trusted and public-available intelligence sources such as Snyk[5] and **National Vulnerability Database**[6] (**NVD**). This data- and knowledge-driven strategy fosters not only users' understanding of how a black-box model works, but also their active engagement in its development and evolution.

Opportunity 3: Combination of Various Explanation Formats. An obvious trend in our analysis was that different explanation formats are commonly used alone. Given their complementarity (e.g., source code explanations and textual explanation are complementary to each other in an AI-assisted programming support context), we believe there is an opportunity to combine diverse formats of explanation for a more robust and complete decision performance, and improves the likelihood of having at least one explanation that the user understands.

Opportunity 4: Human-in-the-Loop Interaction. Exploring a more user-centric approach that provides users with greater agency could lead to results that are orthogonally beneficial to those found using more common techniques. To effectively support human decision-making, there is an escalating need for interactive XAI tools that empower users to actively engage with and explore black-box SE models, thereby facilitating a profound comprehension of the models' mechanisms and their explainability. However, most reviewed works leave aside important aspects pertaining to the XAI tool's interaction with SE practitioners as an AI assistant. Several studies have highlighted that users had more trust when presented with interactive explanations [142]. Thus, one possible research interest could be in the application of human-AI dialogue agent. For instance, LLMs such as ChatGPT can serve as the main controller or "brain" to guide users to clarify their vague intents through multi-round dialogue and return personalized explanations by having access to one or more XAI techniques.

Opportunity 5: Curating High-quality and Multi-dimensional Benchmarks. Evaluating the newly proposed approach over baseline techniques on a benchmark is developing into a standard practice in SE research. Nevertheless, existing benchmarks may face issues related to quality, availability, and personalization. First, ground truth information scarcity is still a major issue since

[5]https://snyk.io/
[6]https://nvd.nist.gov/

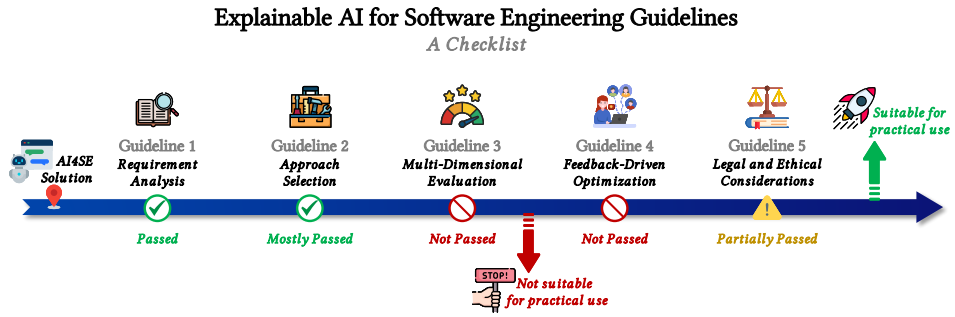


Fig. 11. Guidelines for applying XAI to SE research.

the process of data annotation is expertise-intensive and time-consuming for large-scale datasets. This is even more critical in the XAI field, where additional (and commonly multi-modal) annotations are required (e.g., textual descriptions and structured decision-making rules). A potential solution involves promoting cooperation and collaboration between the industry and academia. We have started to see efforts in constructing high-quality benchmarks with annotated explanatory information in other domains, such as the FFA-IR dataset [62] for evaluating the explainability on medical report generation. Second, the most adopted evaluation approach in current XAI4SE studies is to resort to the practitioners' expertise. However, considering the variability in experts' opinions, this strategy is particularly biased and subjective [91]. Given that such human feedback is immensely valuable in understanding the strengths and weaknesses of XAI techniques, a clear avenue for improvement is to standardize a protocol for human evaluation of these systems by SE practitioners.

7 Guidelines for Future Work on XAI4SE

In this section, we synthesized a checklist with five prescribed steps that should aid in guiding researchers through the process of applying XAI in SE, as illustrated in Figure 11.

Guideline 1: Requirement Analysis. This first step focuses on clarifying specific needs of distinct stakeholders for a certain SE task. For example, for end-users (e.g., developers and practitioners) which do not have technical knowledge in underlying AI models, the format and context of an explanation should be easily understandable so that they can seamlessly incorporate such evidence in their decision process. This involves (❶) determining who uses the model outputs, and in what way; (❷) refining requirements through user surveys; and (❸) standardizing requirements into document form by integrating the feedback.

Guideline 2: Approach Selection. While one of the advantages of out-of-the-box XAI toolkits is that they provide some degree of convenience, it is still important for researchers to carefully consider various aspects of existing explanation techniques and determine whether a given technique could be adapted for their task to prevent the oversimplification of the problem, or whether the creation of a new technique should be considered. As such, this step involves (❶) determining suitable explanation techniques based on task fitness, model compatibility, and stakeholder preference; and (❷) accounting for the combination of different formats of explanation. Additionally, all steps should be thoroughly documented to support replication.

Guideline 3: Multi-Dimensional Evaluation. Explainability is a multi-faceted concept. Thus, this step requires (❶) carefully choosing metrics, especially quantitative measures beyond accuracy, for the task; (❷) testing the approach against well-constructed benchmarks; and (❸) ensuring it has

reached optimum capability. Such a multi-dimensional overview could be implemented as a radar chart that comprehensively and concisely conveys the strengths and weaknesses of the explanation approach. Similar to the previous step, researchers should strive to both include the details of their evaluation plan as well as provide rationale for the choices made.

Guideline 4: Feedback-Driven Optimization. In addition to quantitative evaluation, it is also crucial to investigate the usability of XAI techniques from a human-centric view. This step demonstrates a meaningful attempt to intuitively understand how the proposed approach would perform in a real-world scenario. Here, researchers should integrate XAI4SE solutions into developers' daily development workflows gradually, and conduct empirical studies, such as interviews, questionnaires, and observation of developers using these techniques in real-world scenarios, to gather rich user experiences. Such feedback helps to identify pain points that can further enhance user satisfaction.

Guideline 5: Legal and Ethical Considerations. The final step involves properly evaluating the potential legal and ethical implications before deploying XAI techniques in the wild. Specifically, it is necessary to ensure your data collection process is *compliant*, *privacy-preserving*, and *unbiased* [167, 168]. Moreover, it is also important to carefully consider the possible consequences of inaccurate explanations. Therefore, researchers should take appropriate measures, e.g., conducting audits in a regular manner, to minimize any risks of this kind.

8 Conclusion

Explainability remains a pivotal area of interest within the SE community, particularly as increasingly advanced AI models rapidly advance the field. This article conducts an SLR of 108 primary studies on XAI4SE research from top-tier SE and AI conferences and journals. Initially, we formulated a series of RQs aimed at exploring the application of XAI techniques in SE. Our analysis began by highlighting SE tasks that have significantly benefited from XAI, illustrating the tangible contributions of XAI (RQ1). Subsequently, we delved into the variety of XAI techniques applied to SE tasks, examining their unique characteristics and output formats (RQ2). Following this, we investigated the existing benchmarks, including available baselines, prevalent benchmarks, and commonly employed evaluation metrics, to determine their validity and trustworthiness (RQ3).

Despite the significant contributions made to date, this review also uncovers certain limitations and challenges inherent in existing XAI4SE research, offering a set of guidelines that delineate promising avenues for future exploration. It is our aspiration that this SLR equips future SE researchers with the essential knowledge and insights required for innovative applications of XAI.

References

[1] Shamsa Abid, Xuemeng Cai, and Lingxiao Jiang. 2023. Interpreting CodeBERT for semantic code clone detection. In *Proceedings of the 30th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 229–238.

[2] Jubril Gbolahan Adigun, Tom Philip Huck, Matteo Camilli, and Michael Felderer. 2023. Risk-driven online testing and test case diversity analysis for ML-enabled critical systems. In *Proceedings of the 34th IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 344–354.

[3] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, et al. 2020. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (2020), 82–115.

[4] Md. Abdul Awal and Chanchal K. Roy. 2024. EvaluateXAI: A framework to evaluate the reliability and consistency of rule-based XAI techniques for software analytics tasks. *Journal of Systems and Software* 217 (2024), 112159.

[5] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS One* 10, 7 (2015), e0130140.

[6] Lili Bo, Yuting He, Xiaobing Sun, Wangjie Ji, and Xiaohan Wu. 2024. A software bug fixing approach based on knowledge-enhanced large language models. In *Proceedings of the 24th IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 169–179.

[7] Nadia Burkart and Marco F. Huber. 2021. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research* 70 (2021), 245–317.

[8] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei, and Bin Li. 2021. *BGNN4VD*: Constructing bidirectional graph neural-network for vulnerability detection. *Information and Software Technology* 136 (2021), 106576.

[9] Sicong Cao, Xiaobing Sun, Xiaoxue Wu, David Lo, Lili Bo, Bin Li, and Wei Liu. 2024. Coca: Improving and explaining graph neural network-based vulnerability detection systems. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, 155:1–155:13.

[10] Sicong Cao, Xiaobing Sun, Xiaoxue Wu, David Lo, Lili Bo, Bin Li, Xiaolei Liu, Xingwei Lin, and Wei Liu. 2024. Snopy: Bridging sample denoising with causal graph learning for effective vulnerability detection. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 606–618.

[11] Di Chen, Wei Fu, Rahul Krishna, and Tim Menzies. 2018. Applications of psychological science for actionable analytics. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT)*. ACM, 456–467.

[12] Simin Chen, Zexin Li, Wei Yang, and Cong Liu. 2024. DeciX: Explain deep learning based code generation applications. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2424–2446.

[13] Baijun Cheng, Shengming Zhao, Kailong Wang, Meizhen Wang, Guangdong Bai, Ruitao Feng, Yao Guo, Lei Ma, and Haoyu Wang. 2024. Beyond Fidelity: Explaining Vulnerability Localization of Learning-Based Detectors. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 127:1–127:33.

[14] Shyam R. Chidamber and Chris F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (1994), 476–493.

[15] Zhaoyang Chu, Yao Wan, Qian Li, Yang Wu, Hongyu Zhang, Yulei Sui, Guandong Xu, and Hai Jin. 2024. Graph neural networks for vulnerability detection: A counterfactual explanation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 389–401.

[16] Jürgen Cito, Isil Dillig, Seohyun Kim, Vijayaraghavan Murali, and Satish Chandra. 2021. Explaining mispredictions of machine learning models using rule induction. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 716–727.

[17] Jürgen Cito, Isil Dillig, Vijayaraghavan Murali, and Satish Chandra. 2022. Counterfactual explanations for models of code. In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 125–134.

[18] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (1993), 494.

[19] Fabiano Dalpiaz, Davide Dell'Anna, Fatma Basak Aydemir, and Sercan Çevikol. 2019. Requirements classification with interpretable machine learning and dependency parsing. In *Proceedings of the 27th IEEE International Requirements Engineering Conference (RE)*. IEEE, 142–152.

[20] Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. 2018. Explainable software analytics. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-(NIER)*. ACM, 53–56.

[21] Raphaël Dang-Nhu. 2020. PLANS: Neuro-symbolic program learning from videos. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, 4171–4186.

[23] Jianshu Ding, Guisheng Fan, Huiqun Yu, and Zijie Huang. 2021. Automatic identification of high impact bug report by test smells of textual similar bug reports. In *Proceedings of the 21st IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 446–457.

[24] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, et al. 2023. TraceDiag: Adaptive, interpretable, and efficient root cause analysis on large-scale microservice systems. In *Proceedings of the 31th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 1762–1773.

[25] Yi Ding, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. 2021. Generalizable and interpretable learning for configuration extrapolation. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 728–740.

[26] Geanderson Esteves dos Santos, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and Nivio Ziviani. 2020. Understanding machine learning software defect predictions. *Automated Software Engineering* 27, 3 (2020), 369–392.

[27] Finale Doshi-Velez and Been Kim. 2018. Considerations for evaluation and generalization in interpretable machine learning. *Explainable and Interpretable Models in Computer Vision and Machine Learning* (2018), 3–17.

[28] Finale Doshi-Velez and Been Kim. 2018. Towards A Rigorous Science of Interpretable Machine Learning. 1–13. arXiv:1702.08608. Retrieved from https://arxiv.org/abs/1702.08608

[29] Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Omer F. Rana, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, et al. 2023. Explainable AI (XAI): Core ideas, techniques, and solutions. *ACM Computing Surveys* 55, 9 (2023), 194:1–194:33.

[30] Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Lucas Morales, Luke B. Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. 2021. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*. ACM, 835–850.

[31] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ code vulnerability dataset with code changes and CVE summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. ACM, 508–512.

[32] Benjamin Frész, Elena Dubovitskaya, Danilo Brajovic, Marco F. Huber, and Christian Horz. 2024. How should AI decisions be explained? requirements for explanations from the perspective of european law. In *Proceedings of the 7th AAAI/ACM Conference on AI, Ethics, and Society (AIES)*. AAAI, 438–450.

[33] Jerome H. Friedman and Bogdan E. Popescu. 2008. Predictive learning via rule ensembles. *The Annals of Applied Statistics* 2, 3 (2008), 916–954.

[34] Michael Fu, Van Nguyen, Chakkrit Tantithamthavorn, Trung Le, and Dinh Phung. 2023. VulExplainer: A transformer-based hierarchical distillation for explaining vulnerability types. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4550–4565.

[35] Michael Fu and Chakkrit Tantithamthavorn. 2023. GPT2SP: A transformer-based agile story point estimation approach. *IEEE Transactions on Software Engineering* 49, 2 (2023), 611–625.

[36] Yuxiang Gao, Yi Zhu, and Qiao Yu. 2022. Evaluating the effectiveness of local explanation methods on source code-based defect prediction models. In *Proceedings of the 19th IEEE/ACM International Conference on Mining Software Repositories (MSR)*. ACM, 640–645.

[37] Yuxiang Gao, Yi Zhu, and Yu Zhao. 2022. Dealing with imbalanced data for interpretable defect prediction. *Information and Software Technology* 151 (2022), 107016.

[38] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Shaomeng Cao, Kechi Zhang, and Zhi Jin. 2023. Interpretation-based code summarization. In *Proceedings of the 31st IEEE/ACM International Conference on Program Comprehension (ICPC)*. IEEE, 113–124.

[39] Jiri Gesi, Xinyun Shen, Yunfan Geng, Qihong Chen, and Iftekhar Ahmed. 2023. Leveraging feature bias for scalable misprediction explanation of machine learning models. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 1559–1570.

[40] Amirata Ghorbani, Abubakar Abid, and James Y. Zou. 2019. Interpretation of neural networks is fragile. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 3681–3688.

[41] Bryce Goodman and Seth R. Flaxman. 2017. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine* 38, 3 (2017), 50–57.

[42] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A survey of methods for explaining black box models. *ACM Computing Surveys* 51, 5 (2019), 93:1–93:42.

[43] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. LEMNA: Explaining deep learning based security applications. In *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 364–379.

[44] Jianjun He, Ling Xu, Yuanrui Fan, Zhou Xu, Meng Yan, and Yan Lei. 2020. Deep learning based valid bug reports determination and explanation. In *Proceedings of the 31st IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 184–194.

[45] Adha Hrusto, Per Runeson, and Magnus C. Ohlsson. 2024. Autonomous monitors for detecting failures early and reporting interpretable alerts in cloud operations. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, 47–57.

[46] Yutao Hu, Suyuan Wang, Wenke Li, Junru Peng, Yueming Wu, Deqing Zou, and Hai Jin. 2023. Interpreters for GNN-based vulnerability detection: Are we there yet?. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 1407–1419.

[47] Qing Huang, Zishuai Li, Zhenchang Xing, Zhengkang Zuo, Xin Peng, Xiwei Xu, and Qinghua Lu. 2024. Answering uncertain, under-specified API queries assisted by knowledge-aware human-AI dialogue. *IEEE Transactions on Software Engineering* 50, 2 (2024), 280–295.

[48] Zijie Huang, Huiqun Yu, Guisheng Fan, Zhiqing Shao, Ziyi Zhou, and Mingchen Li. 2024. On the effectiveness of developer features in code smell prioritization: A replication study. *Journal of Systems and Software* 210 (2024), 111968.
[49] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. 1–6. arXiv:1909.09436. Retrieved from https://arxiv.org/abs/1909.09436
[50] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. CURE: Code-aware neural machine translation for automatic program repair. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 1161–1173.
[51] Shuyao Jiang, Jiacheng Shen, Shengnan Wu, Yu Cai, Yue Yu, and Yangfan Zhou. 2023. Towards usable neural comment generation via code-comment linkage interpretation: Method and empirical study. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2239–2254.
[52] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and John C. Grundy. 2021. Practitioners' perceptions of the goals and visual explanations of defect prediction models. In *Proceedings of the 18th IEEE/ACM International Conference on Mining Software Repositories (MSR)*. IEEE, 432–443.
[53] Jirayus Jiarpakdee, Chakkrit Kla Tantithamthavorn, Hoa Khanh Dam, and John C. Grundy. 2022. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering* 48, 2 (2022), 166–185.
[54] Baharin Aliashrafi Jodat, Abhishek Chandar, Shiva Nejati, and Mehrdad Sabetzadeh. 2024. Test generation strategies for building failure models and explaining spurious failures. *ACM Transactions on Software Engineering and Methodology* 33, 4 (2024), 93:1–93:32.
[55] Alexander Kampmann, Nikolas Havrikov, Ezekiel O. Soremekun, and Andreas Zeller. 2020. When does my program do this? learning circumstances of software behavior. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 1228–1239.
[56] Sungmin Kang, Gabin An, and Shin Yoo. 2024. A quantitative and qualitative evaluation of llm-based explainable fault localization. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1424–1446.
[57] Wenjun Ke, Chao Wu, Xiufeng Fu, Chen Gao, and Yinyi Song. 2020. Interpretable test case recommendation based on knowledge graph. In *Proceedings of the 20th IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 489–496.
[58] Bonan Kou, Shengmai Chen, Zhijie Wang, Lei Ma, and Tianyi Zhang. 2024. Do large language models pay similar attention like human programmers when generating code? *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2261–2284.
[59] Muhammad Laiq, Nauman Bin Ali, Jürgen Börstler, and Emelie Engström. 2024. Industrial adoption of machine learning techniques for early identification of invalid bug reports. *Empirical Software Engineering* 29, 5 (2024), 130.
[60] Johannes Lampel, Sascha Just, Sven Apel, and Andreas Zeller. 2021. When life gives you oranges: Detecting and diagnosing intermittent job failures at mozilla. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 1381–1392.
[61] Gichan Lee and Scott Uk-Jin Lee. 2023. An empirical comparison of model-agnostic techniques for defect prediction models. In *Proceedings of the 30th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 179–189.
[62] Mingjie Li, Wenjia Cai, Rui Liu, Yuetian Weng, Xiaoyun Zhao, Cong Wang, Xin Chen, Zhong Liu, Caineng Pan, Mengke Li, et al. 2021. FFA-IR: Towards an explainable and reliable medical report generation benchmark. In *Proceedings of the 1st Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks)*.
[63] Rongfan Li, Bihuan Chen, Fengyi Zhang, Chao Sun, and Xin Peng. 2022. Detecting runtime exceptions by deep code representation learning with attention-based graph neural networks. In *Proceedings of the 29th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 373–384.
[64] Yangguang Li, Zhen Ming (Jack) Jiang, Heng Li, Ahmed E. Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. 2020. Predicting node failures in an ultra-large-scale cloud computing platform: An aiops solution. *ACM Transactions on Software Engineering and Methodology* 29, 2 (2020), 13:1–13:24.
[65] Yi Li, Shaohua Wang, and Tien N. Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceeding of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 292–303.
[66] Zhen Li, Ruqian Zhang, Deqing Zou, Ning Wang, Yating Li, Shouhuai Xu, Chen Chen, and Hai Jin. 2023. Robin: A Novel Method to Produce Robust Interpreters for Deep Learning-Based Code Classifiers. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 27–39.
[67] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 996–1008.

[68] Yifan Liao, Ming Xu, Yun Lin, Xiwen Teoh, Xiaofei Xie, Ruitao Feng, Frank Liauw, Hongyu Zhang, and Jin Song Dong. 2024. Detecting and explaining anomalies caused by web tamper attacks via building consistency-based normality. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 531–543.

[69] Dayi Lin, Chakkrit Tantithamthavorn, and Ahmed E. Hassan. 2022. The impact of data merging on the interpretation of cross-project just-in-time defect models. *IEEE Transactions on Software Engineering* 48, 8 (2022), 2969–2986.

[70] Hui Liu, Mingzhu Shen, Jiaqi Zhu, Nan Niu, Ge Li, and Lu Zhang. 2022. Deep learning based program generation from requirements text: Are we there yet? *IEEE Transactions on Software Engineering* 48, 4 (2022), 1268–1289.

[71] Mingwei Liu, Simin Yu, Xin Peng, Xueyin Du, Tianyong Yang, Huanjun Xu, and Gaoyang Zhang. 2023. Knowledge graph based explainable question retrieval for programming tasks. In *Proceedings of the 39th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 123–135.

[72] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. 2022. Explainable AI for android malware detection: Towards understanding why the models perform so well?. In *Proceedings of the IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 169–180.

[73] Yue Liu, Chakkrit Tantithamthavorn, Yonghui Liu, and Li Li. 2024. On the reliability and explainability of language models for program generation. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 126:1–126:26.

[74] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yuhang Chen, Yanqing Zhao, Hao Yang, and Yanfei Jiang. 2024. Interpretable online log analysis using large language models with prompt strategies. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC)*. ACM, 35–46.

[75] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. 2021. Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*. 2751–2759.

[76] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021*.

[77] Scott M. Lundberg, Gabriel G. Erion, Hugh Chen, Alex J. DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence* 2, 1 (2020), 56–67.

[78] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*. 4765–4774.

[79] Yingzhe Lyu, Gopi Krishnan Rajbahadur, Dayi Lin, Boyuan Chen, and Zhen Ming (Jack) Jiang. 2022. Towards a Consistent Interpretation of AIOps Models. *ACM Transactions on Software Engineering and Methodology* 31, 1 (2022), 16:1–16:38.

[80] Lezhi Ma, Shangqing Liu, Yi Li, Xiaofei Xie, and Lei Bu. 2025. SpecGen: Automated generation of formal program specifications via large language models. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE.

[81] Wei Ma, Shangqing Liu, Mengjie Zhao, Xiaofei Xie, Wenhan Wang, Qiang Hu, Jie Zhang, and Yang Liu. 2024. Unveiling code pre-trained models: Investigating syntax and semantics capacities. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 169:1–169:29.

[82] Parvez Mahbub, Ohiduzzaman Shuvo, and Mohammad Masudur Rahman. 2023. Explaining software bugs leveraging code structures in neural machine translation. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 640–652.

[83] Vadim Markovtsev, Waren Long, Hugo Mougard, Konstantin Slavnov, and Egor Bulychev. 2019. STYLE-ANALYZER: Fixing code style inconsistencies with interpretable unsupervised algorithms. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*. IEEE/ACM, 468–478.

[84] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Proceedings of the 36th Annual Conference on Neural Information Processing Systems (NeurIPS)*.

[85] Ahmad Haji Mohammadkhani, Nitin Sai Bommi, Mariem Daboussi, Onkar Sabnis, Chakkrit Tantithamthavorn, and Hadi Hemmati. 2023. A systematic literature review of explainable AI for software engineering. 1–23. arXiv:2302.06065. Retrieved from https://arxiv.org/abs/2302.06065

[86] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition* 65 (2017), 211–222.

[87] Toshiki Mori and Naoshi Uchihira. 2019. Balancing the tradeoff between accuracy and interpretability in software defect prediction. *Empirical Software Engineering* 24, 2 (2019), 779–825.

[88] Tamara Munzner. 2014. *Visualization Analysis and Design*. CRC Press.

[89] Azqa Nadeem, Daniël Vos, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. 2023. SoK: Explainable machine learning for computer security applications. In *Proceedings of the 8th IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 221–240.

[90] David Nader-Palacio, Alejandro Velasco, Nathan Cooper, Alvaro Rodriguez, Kevin Moran, and Denys Poshyvanyk. 2024. Toward a theory of causation for interpreting neural code models. *IEEE Transactions on Software Engineering* 50, 5 (2024), 1215–1243.

[91] Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. 2023. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable AI. *ACM Computing Surveys* 55, 13s (2023), 295:1–295:42.

[92] Amirmohammad Nazari, Yifei Huang, Roopsha Samanta, Arjun Radhakrishna, and Mukund Raghothaman. 2023. Explainable program synthesis by localizing specifications. In *Proceedings of the 38th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 2171–2195.

[93] Chao Ni, Xin Yin, Kaiwen Yang, Dehai Zhao, Zhenchang Xing, and Xin Xia. 2023. Distinguishing look-alike innocent and vulnerable code by subtle semantic representation learning and explanation. In *Proceedings of the 31th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 1611–1622.

[94] Marc North, Amir Atapour-Abarghouei, and Nelly Bencomo. 2024. Code gradients: Towards automated traceability of llm-generated code. In *Proceedings of the 32nd IEEE International Requirements Engineering Conference (RE)*. IEEE, 321–329.

[95] Ipek Ozkaya and James Ivers. 2019. AI for software engineering. In *Proceedings of the SEI Educator's Workshop*.

[96] David N. Palacio, Dipin Khati, Daniel Rodríguez-Cárdenas, Alejandro Velasco, and Denys Poshyvanyk. 2025. On explaining (large) language models for code using global code-based explanations. 1–12. arXiv:2503.16771. Retrieved from https://arxiv.org/abs/2503.16771

[97] Matteo Paltenghi, Rahul Pandita, Austin Z. Henley, and Albert Ziegler. 2024. Follow-up attention: An empirical study of developer and neural model code exploration. *IEEE Transactions on Software Engineering* 50, 10 (2024), 2568–2582.

[98] Matteo Paltenghi and Michael Pradel. 2021. Thinking like a developer? comparing the attention of humans with neural models of code. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 867–879.

[99] Cristiano Patrício, João C. Neve, and Luís F. Teixeira. 2023. Explainable deep learning methods in medical image classification: A survey. *ACM Computing Surveys* 56, 4 (2023), 85:1–85:41.

[100] Fabiano Pecorelli, Fabio Palomba, Foutse Khomh, and Andrea De Lucia. 2020. Developer-driven code smell prioritization. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. ACM, 220–231.

[101] Naser Peiravian and Xingquan Zhu. 2013. Machine learning for android malware detection using permission and API calls. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE Computer Society, 300–305.

[102] Kewen Peng and Tim Menzies. 2022. Defect reduction planning (using timelime). *IEEE Transactions on Software Engineering* 48, 7 (2022), 2510–2525.

[103] Chanathip Pornprasit, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Michael Fu, and Patanamon Thongtanunam. 2021. PyExplainer: Explaining the predictions of just-in-time defect models. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 407–418.

[104] Md. Rafiqul Islam Rabin, Vincent J. Hellendoorn, and Mohammad Amin Alipour. 2021. Understanding neural code intelligence through program simplification. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 441–452.

[105] Dilini Rajapaksha, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Christoph Bergmeir, John Grundy, and Wray L. Buntine. 2022. SQAPlanner: Generating data-informed software quality improvement plans. *IEEE Transactions on Software Engineering* 48, 8 (2022), 2814–2835.

[106] Rui Ren, Jinbang Yang, Linxiao Yang, Xinyue Gu, and Liang Sun. 2024. SLIM: A scalable and interpretable light-weight fault localization algorithm for imbalanced data in microservice. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 27–39.

[107] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. 2019. Neural network-based detection of self-admitted technical debt: From performance to explainability. *ACM Transactions on Software Engineering and Methodology* 28, 3 (2019), 15.

[108] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "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 (KDD)*. ACM, 1135–1144.

[109] Daniel Rodríguez-Cárdenas, David N. Palacio, Dipin Khati, Henry Burke, and Denys Poshyvanyk. 2023. Benchmarking causal study to interpret large language models for source code. In *Proceedings of the 39th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 329–334.

[110] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (2019), 206–215.

[111] Cynthia Rudin and Joanna Radin. 2019. Why are we using black box models in AI when we don't need to? a lesson from an explainable AI competition. *Harvard Data Science Review* 1, 2 (2019), 1–9.

[112] Nayan B. Ruparelia. 2010. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes* 35, 3 (2010), 8–13.

[113] Jaydeb Sarker, Sayma Sultana, Steven R. Wilson, and Amiangshu Bosu. 2023. ToxiSpanSE: An explainable toxicity detection in code review comments. In *Proceedings of the 17th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[114] Lukas Schulte, Benjamin Ledel, and Steffen Herbold. 2024. Studying the explanations for the automated prediction of bug and non-bug issues using LIME and SHAP. *Empirical Software Engineering* 29, 4 (2024), 93.

[115] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2020. Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision* 128, 2 (2020), 336–359.

[116] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken-ichi Matsumoto. 2013. Studying re-opened bugs in open source software. *Empirical Software Engineering* 18, 5 (2013), 1005–1042.

[117] Jiho Shin, Reem Aleithan, Jaechang Nam, Junjie Wang, Nima Shiri Harzevili, and Song Wang. 2023. An empirical study on the stability of explainable software defect prediction. In *Proceedings of the 30th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 141–150.

[118] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. PMLR, 3145–3153.

[119] Timo Speith. 2022. A review of taxonomies of explainable artificial intelligence (XAI) methods. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*. 2239–2250.

[120] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. 2020. explAIner: A visual analytics framework for interactive and explainable machine learning. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2020), 1064–1074.

[121] M. Srinivas and Lalit M. Patnaik. 1994. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics* 24, 4 (1994), 656–667.

[122] Mateusz Staniak and Przemyslaw Biecek. 2018. Explanations of model predictions with live and breakdown packages. *R J.* 10, 2 (2018), 395.

[123] Jiamou Sun, Zhenchang Xing, Qinghua Lu, Xiwei Xu, Liming Zhu, Thong Hoang, and Dehai Zhao. 2023. Silent vulnerable dependency alert prediction with vulnerability key aspect explanation. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 970–982.

[124] Ruoxi Sun, Minhui Xue, Gareth Tyson, Tian Dong, Shaofeng Li, Shuo Wang, Haojin Zhu, Seyit Camtepe, and Surya Nepal. 2023. Mate! are you really aware? an explainability-guided testing framework for robustness of malware detectors. In *Proceedings of the 31th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 1573–1585.

[125] Yongqian Sun, Zihan Lin, Binpeng Shi, Shenglin Zhang, Shiyu Ma, Pengxiang Jin, Zhenyu Zhong, Lemeng Pan, Yicheng Guo, and Dan Pei. 2024. Interpretable failure localization for microservice systems based on graph autoencoder. *ACM Transactions on Software Engineering and Methodology* 34, 2 (2025), 52:1–52:28.

[126] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*. PMLR, 3319–3328.

[127] Sahil Suneja, Yufan Zhuang, Yunhui Zheng, Jim Laredo, Alessandro Morari, and Udayan Khurana. 2023. Incorporating signal awareness in source code modeling: An application to vulnerability detection. *ACM Transactions on Software Engineering and Methodology* 32, 6 (2023), 145:1–145:40.

[128] Ben Tang, Bin Li, Lili Bo, Xiaoxue Wu, Sicong Cao, and Xiaobing Sun. 2021. GrasP: Graph-to-sequence learning for automated program repair. In *Proceedings of the 21st IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 819–828.

[129] Dimitrios Tsoukalas, Nikolaos Mittas, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Dionysios D. Kehagias. 2024. Local and global explainability for technical debt identification. *IEEE Transactions on Software Engineering* 50, 8 (2024), 2110–2123.

[130] Michael van Lent, William Fisher, and Michael Mancuso. 2004. An explainable artificial intelligence system for small-unit tactical behavior. In *Proceedings of the 19th National Conference on Artificial Intelligence, 16th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*. AAAI/ The MIT, 900–907.

[131] Alejandro Velasco, David N. Palacio, Daniel Rodríguez-Cárdenas, and Denys Poshyvanyk. 2024. Which syntactic capabilities are statistically learned by masked language models for code?. In *Proceedings of the 44th ACM/IEEE International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NEIR)*. ACM, 72–76.

[132] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

[133] Yao Wan, Jingdong Shu, Yulei Sui, Guandong Xu, Zhou Zhao, Jian Wu, and Philip S. Yu. 2019. Multi-modal attention network learning for semantic source code retrieval. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 13–25.

[134] Yao Wan, Wei Zhao, Hongyu Zhang, Yulei Sui, Guandong Xu, and Hai Jin. 2022. What do they capture? - A structural analysis of pre-trained language models for source code. In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, 2377–2388.

[135] Chong Wang, Xin Peng, Zhenchang Xing, Yue Zhang, Mingwei Liu, Rong Luo, and Xiujie Meng. 2023. XCoS: Explainable code search based on query scoping and knowledge graph. *ACM Transactions on Software Engineering and Methodology* 32, 6 (2023), 140:1–140:28.

[136] Haoye Wang, Zhipeng Gao, Xing Hu, David Lo, John Grundy, and Xinyu Wang. 2024. Just-in-time todo-missed commits detection. *IEEE Transactions on Software Engineering* 50, 11 (2024), 2732–2752.

[137] Qinqin Wang, Hanbing Yan, and Zhihui Han. 2021. Explainable APT attribution for malware using NLP techniques. In *Proceedings of the 21st IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 70–80.

[138] Xin Wang, Jin Liu, Li Li, Xiao Chen, Xiao Liu, and Hao Wu. 2020. Detecting and explaining self-admitted technical debts with attention-based neural networks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 871–882.

[139] Yu Wang, Ke Wang, and Linzhang Wang. 2023. An explanation method for models of code. In *Proceedings of the 38th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 801–827.

[140] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Hideaki Hata, and Kenichi Matsumoto. 2022. Predicting defective lines using a model-agnostic technique. *IEEE Transactions on Software Engineering* 48, 5 (2022), 1480–1496.

[141] William Webber, Alistair Moffat, and Justin Zobel. 2010. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems* 28, 4 (2010), 20:1–20:38.

[142] Katharina Weitz, Dominik Schiller, Ruben Schlagowski, Tobias Huber, and Elisabeth André. 2019. "Do you trust me?": Increasing user-trust by integrating virtual agents in explainable AI interaction design. In *Proceedings of the 19th ACM International Conference on Intelligent Virtual Agents (IVA)*. ACM, 7–9.

[143] Ratnadira Widyasari, Jia Wei Ang, Truong Giang Nguyen, Neil Sharma, and David Lo. 2024. Demystifying faulty code: Step-by-step reasoning for explainable fault localization. In *Proceedings of the 31st IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 568–579.

[144] Ratnadira Widyasari, Gede Artha Azriadi Prana, Stefanus A. Haryono, Yuan Tian, Hafil Noer Zachiary, and David Lo. 2022. XAI4FL: Enhancing spectrum-based fault localization with explainable artificial intelligence. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC)*. ACM, 499–510.

[145] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Proceedings of the Breakthroughs in Statistics: Methodology and Distribution*. Springer, 196–202.

[146] Jan De Winter. 2010. Explanations in software engineering: The pragmatic point of view. *Minds and Machines* 20, 2 (2010), 277–289.

[147] Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R. Lyu. 2021. Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology* 30, 2 (2021), 21:1–21:29.

[148] Chunqiu Steven Xia and Lingming Zhang. 2024. Automated program repair via conversation: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 819–831.

[149] Wenxin Xiao, Hao He, Weiwei Xu, Yuxia Zhang, and Minghui Zhou. 2023. How early participation determines long-term sustained activity in github projects?. In *Proceedings of the 31th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 29–41.

[150] Xinqiang Xie, Xiaochun Yang, Bin Wang, and Qiang He. 2022. DevRec: Multi-relationship embedded software developer recommendation. *IEEE Transactions on Software Engineering* 48, 11 (2022), 4357–4379.

[151] Fengyu Yang, Guangdong Zeng, Fa Zhong, Peng Xiao, Wei Zheng, and Fuxing Qiu. 2024. CfExplainer: Explainable just-in-time defect prediction based on counterfactuals. *Journal of Systems and Software* 218 (2024), 112182.

[152] Fengyu Yang, Guangdong Zeng, Fa Zhong, Wei Zheng, and Peng Xiao. 2023. Interpretable software defect prediction incorporating multiple rules. In *Proceedings of the 30th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 940–947.

[153] Jia Yang, Cai Fu, Fengyang Deng, Ming Wen, Xiaowei Guo, and Chuanhao Wan. 2023. Toward interpretable graph tensor convolution neural network for code semantics embedding. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 115:1–115:40.

[154] Lanxin Yang, Jinwei Xu, Yifan Zhang, He Zhang, and Alberto Bacchelli. 2023. EvaCRC: Evaluating code review comments. In *Proceedings of the 31th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 275–287.

[155] Zhou Yang, Zhensu Sun, Terry Zhuo Yue, Premkumar Devanbu, and David Lo. 2024. Robustness, security, privacy, explainability, efficiency, and usability of large language models for code. 1–35. arXiv:2403.07506. Retrieved from https://arxiv.org/abs/2403.07506

[156] Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. 2024. Chain-of-event: Interpretable root cause analysis for microservices through automatically learning weighted event causal graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*. ACM, 50–61.

[157] Suraj Yatish, Jirayus Jiarpakdee, Patanamon Thongtanunam, and Chakkrit Tantithamthavorn. 2019. Mining Software Defects: Should We Consider Affected Releases?. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. /ACM, 654–665.

[158] Wei Jie Yeo, Wihan van der Heever, Rui Mao, Erik Cambria, Ranjan Satapathy, and Gianmarco Mengaldo. 2023. A comprehensive review on financial explainable AI. *Artificial Intelligence Review* 58, 6 (2025), 189.

[159] Xin Yin, Chongyang Shi, and Shuxin Zhao. 2021. Local and global feature based explainable feature envy detection. In *Proceedings of the IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 942–951.

[160] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: Generating explanations for graph neural networks. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*. 9240–9251.

[161] Hao Yu, Yiling Lou, Ke Sun, Dezhi Ran, Tao Xie, Dan Hao, Ying Li, Ge Li, and Qianxiang Wang. 2022. Automated assertion generation via information retrieval and its integration with deep learning. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. ACM, 163–174.

[162] Jinqiang Yu, Michael Fu, Alexey Ignatiev, Chakkrit Tantithamthavorn, and Peter Stuckey. 2024. A formal explainer for just-in-time defect predictions. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 187:1–187:31.

[163] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering* 28, 2 (2002), 183–200.

[164] Zhengran Zeng, Yuqun Zhang, Yong Xu, Minghua Ma, Bo Qiao, Wentao Zou, Qingjun Chen, Meng Zhang, Xu Zhang, Hongyu Zhang, et al. 2023. TraceArk: Towards actionable performance anomaly alerting for online service systems. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 258–269.

[165] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Information and Software Technology* 53, 6 (2011), 625–637.

[166] Jian Zhang, Shangqing Liu, Xu Wang, Tianlin Li, and Yang Liu. 2023. Learning to locate and describe vulnerabilities. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 332–344.

[167] Jiale Zhang, Chengcheng Zhu, Chunpeng Ge, Chuan Ma, Yanchao Zhao, Xiaobing Sun, and Bing Chen. 2024. BadCleaner: Defending backdoor attacks in federated learning via attention-based multi-teacher distillation. *IEEE Transactions on Dependable and Secure Computing* 21, 5 (2024), 4559–4573.

[168] Jiale Zhang, Chengcheng Zhu, Xiaobing Sun, Chunpeng Ge, Bing Chen, Willy Susilo, and Shui Yu. 2024. FLPurifier: Backdoor defense in federated learning via decoupled contrastive training. *IEEE Transactions on Information Forensics and Security* 19 (2024), 4752–4766.

[169] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, and David Lo. 2025. Revisiting sentiment analysis for software engineering in the era of large language models. *ACM Transactions on Software Engineering and Methodology* 34, 3 (2025), 60:1–60:30.

[170] Zhuo Zhang, Yan Lei, Meng Yan, Yue Yu, Jiachi Chen, Shangwen Wang, and Xiaoguang Mao. 2022. Reentrancy vulnerability detection and localization: A deep learning based two-phase approach. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 83:1–83:13.

[171] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology* 15, 2 (2024), 20:1–20:38.

[172] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, et al. 2020. Real-time incident prediction for online service systems. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 315–326.

[173] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, et al. 2021. An empirical Investigation of Practical Log Anomaly Detection for Online Service Systems. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 1404–1415.

[174] Wei Zheng, Tianren Shen, Xiang Chen, and Peiran Deng. 2022. Interpretability application of the just-in-time software defect prediction model. *Journal of Systems and Software* 188 (2022), 111245.

[175] Jiayuan Zhou, Michael Pacheco, Jinfu Chen, Xing Hu, Xin Xia, David Lo, and Ahmed E. Hassan. 2023. CoLeFunDa: Explainable silent vulnerability fix identification. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2565–2577.

[176] Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*. 10197–10207.

[177] Deqing Zou, Yawei Zhu, Shouhuai Xu, Zhen Li, Hai Jin, and Hengkai Ye. 2021. Interpreting deep learning-based vulnerability detector predictions based on heuristic searching. *ACM Transactions on Software Engineering and Methodology* 30, 2 (2021), 23:1–23:31.

Received 26 January 2024; revised 3 June 2025; accepted 29 July 2025