

# A Systematic Literature Review on Explainability for Machine/Deep Learning-based Software Engineering Research

SICONG CAO, Yangzhou University, China

XIAOBING SUN\*, Yangzhou University, China

RATNADIRA WIDYASARI, Singapore Management University, Singapore

DAVID LO, Singapore Management University, Singapore

XIAOXUE WU, Yangzhou University, China

LILI BO, Yangzhou University, China

JIALE ZHANG, Yangzhou University, China

BIN LI, Yangzhou University, China

WEI LIU, Yangzhou University, China

DI WU, University of Southern Queensland, Australia

YIXIN CHEN, Washington University in St. Louis, USA

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 paper 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 & AI conferences and journals, and spans 63 papers across 21 unique SE tasks. Based on three key Research Questions (RQs), we aim to (1) summarize the SE tasks 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 roadmap highlighting potential opportunities we deemed appropriate and important for future work.

---

\*Corresponding author

Authors' addresses: Sicong Cao, School of Information Engineering, Yangzhou University, Yangzhou, China, DX120210088@yzu.edu.cn; Xiaobing Sun, School of Information Engineering, Yangzhou University, Yangzhou, China, xbsun@yzu.edu.cn; Ratnadira Widyasari, School of Computing and Information Systems, Singapore Management University, Singapore, ratnadiraw.2020@phdcs.smu.edu.sg; David Lo, School of Computing and Information Systems, Singapore Management University, Singapore, davidlo@smu.edu.sg; Xiaoxue Wu, School of Information Engineering, Yangzhou University, Yangzhou, China, xiaoxuewu@yzu.edu.cn; Lili Bo, School of Information Engineering, Yangzhou University, Yangzhou, China, lilibo@yzu.edu.cn; Jiale Zhang, School of Information Engineering, Yangzhou University, Yangzhou, China, jialezhang@yzu.edu.cn; Bin Li, School of Information Engineering, Yangzhou University, Yangzhou, China, lb@yzu.edu.cn; Wei Liu, School of Information Engineering, Yangzhou University, Yangzhou, China, weiliu@yzu.edu.cn; Di Wu, School of Mathematics, Physics, and Computing, University of Southern Queensland, Toowoomba, Australia, di.wu@unisq.edu.au; Yixin Chen, Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, USA, chen@cse.wustl.edu.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Association for Computing Machinery.

Manuscript submitted to ACM

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 Widayasi, David Lo, Xiaoxue Wu, Lili Bo, Jiale Zhang, Bin Li, Wei Liu, Di Wu, and Yixin Chen. 2023. A Systematic Literature Review on Explainability for Machine/Deep Learning-based Software Engineering Research. 1, 1 (January 2023), 41 pages. <https://doi.org/XXXXXX.XXXXXXX>

## 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 unprecedently 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 [38, 56], code generation [49], and clone detection [105, 109]. 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 [70].

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 [32]. 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 [68, 129]. For the model designers, explainability is required to investigate failure cases and direct the weak AI models in the proper paths as intended [93]. 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)** [20].

**Related Surveys.** Substantial efforts have been underway in recent years to enhance the explainability of black-box models. These approaches operate either by analyzing the impact of specific regions of the input on the final prediction [55, 78], or by inspecting the network activation [83, 134]. Table 1 summarises some of their successful applications in downstream fields, such as healthcare [59] and finance [121]. Simultaneously, the SE community also embarks on a series of research activities regarding explainability. This effort has led to advancements across a range of tasks including defect prediction [57, 74], vulnerability detection [36, 137], code summarization [30, 39], malware detection [51, 113], among many others [9, 50, 103]. Recently, Mohammadkhani et al. [61] 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. However, their survey still has several limitations. First, they only collected and analyzed 24 papers published before June 2022. There is a need to investigate a larger collection of papers, including more recent ones. Second,

Table 1. Comparison of Our Work with Previous Surveys/Reviews on Explainability for Downstream Applications

Reference	Year	Application Scenario	#Papers	Taxonomy	Baseline	Benchmark	Metric
Patrício et al. [73]	2023	Healthcare	53	Customized	●	●	●
Messina et al. [59]	2022	Healthcare	40	Hybrid	●	●	●
Hossain et al. [35]	2023	Healthcare	218	Hybrid	○	○	●
Zini et al. [136]	2022	Natural Language Processing	-	General	○	○	○
Jie et al. [121]	2023	Finance	69	General	○	○	○
Zablocki et al. [125]	2022	Autonomous Driving	-	General	○	○	●
Mohammadhkhani et al. [61]	2023	Software Engineering	24	General	○	○	○
<b>Our survey</b>	2024	Software Engineering	63	Customized	●	●	●

they directly borrowed the general taxonomy, i.e., ante- and post-hoc explanation, from the AI field to classify XAI techniques in SE tasks. This taxonomy is coarse-grained and may not be applicable to specific downstream applications. Third, they did not (or only partially) explicitly discuss the evaluation aspects of reviewed papers, 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.

**Our Contributions.** To effectively chart the most promising path forward for research on the utilization of XAI techniques in AI4SE, we conducted a **Systematic Literature Review (SLR)** to bridge this gap, providing valuable insights to the community. In this paper, we collected **63** primary studies published in 26 flagship conferences and journals over the last 12 years (2012-2023). We then summarized the research topics tackled by relevant papers, classified various XAI techniques used in diverse SE tasks, and analyzed the evaluation process. 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 63 primary studies on the topic of explainability for machine/deep learning-based software engineering research, for researchers and practitioners interested in prioritizing transparency in their solutions.
- We describe the key applications of XAI4SE encompassing a diverse range of 21 specific SE tasks, grouped into four core SE activities, including software development, software testing, software maintenance, and software management.
- We provide a taxonomy of XAI techniques used in SE based on their methodology, and an analysis of frequently used explanation formats.
- We summarize existing benchmarks used in XAI4SE approaches, including available baselines, prevalent benchmarks, and commonly employed evaluation metrics, to determine their validity.
- We discuss key challenges that using XAI encounters within the SE field and suggest several potential research directions for XAI4SE.
- We maintain an interactive website, <https://riss-vul.github.io/xai4se-paper/>, with all of our data and results to facilitate reproducibility and encourage contributions from the community to continue to push forward XAI4SE research.

**Paper Organization.** The remainder of this paper is structured as follows: Section 2 describes the preliminaries of XAI. Section 3 presents **Research Questions (RQs)** and our SLR methodology. The succeeding Sections 4-6 are devoted to answering each of these RQs individually. Section 7 presents the limitations of this study. Section 8 discusses the

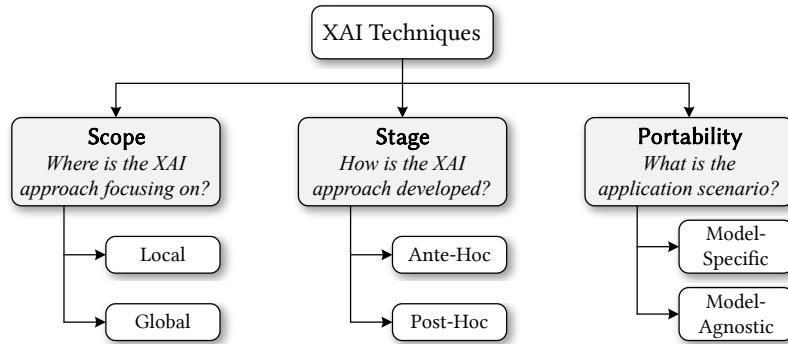


Fig. 1. General taxonomy of the survey in terms of scope, stage, and portability.

challenges that still need to be solved in future work and outlines a clear research roadmap of research opportunities. Section 9 concludes this paper.

## 2 EXPLAINABLE ARTIFICIAL INTELLIGENCE: PRELIMINARIES

This section first details the definitions of 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 [5, 65]. For example, quoting Doshi-Velez and Kim et al. [19], interpretability is the ability “to explain or to present in understandable terms to a human.” By contrast, according to Lent et al. [95], 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, 66]. 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. [11] 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: (1) making the entire decision-making process transparent and comprehensible (i.e., white-box/interpretable models); and (2) 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 [111]. As we observed in Section 5.2, 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 XAI field. Based on the previous literature, most XAI approaches can be categorized according to three criteria [86]: (1) *scope* (local vs. global); (2) *stage* (ante-hoc vs. post-hoc); (3) and *portability* (model-specific vs. model-agnostic). Fig. 1 illustrates the general taxonomy of the XAI techniques, and each category is detailed in the following.

**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 seek to explain why a model performs a specific prediction for an individual input. For example, given a file of interest and a defect prediction model, a locally explainable model might generate attribution scores for each code token in the file [107]. Representative local explainability approaches include LIME [78], SHAP [55], and Grad-CAM [83]. Global explainability approaches work on an array of inputs to give insights into the overall behavior of the black-box model. Various rule-based and tree-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* in certain literature), while the latter refers to *post-hoc explainability*. By definition, most inherently interpretable approaches are model-specific such that any change in the architecture will need significant changes in the approach itself. On the other hand, significant research interest in recent years is seen in developing post-hoc explanations as they can explain a well-trained black-box model decision without sacrificing the accuracy. 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. For example, Deconvolution [126] is model-specific due to its ability to only explain the CNN model. Conversely, model-agnostic approaches can be used to explain arbitrary models without being constrained to any particular model architecture.

## 3 REVIEW METHODOLOGY

### 3.1 Research Question

In this paper, we focus on investigating the following **Research Questions (RQs)**:

- **RQ<sub>1</sub>: What types of AI4SE studies have been explored for explainability?**
- **RQ<sub>2</sub>: How XAI techniques are used to support SE tasks?**
  - **RQ<sub>2a</sub>: What types of XAI techniques are employed to generate explanations?**
  - **RQ<sub>2b</sub>: What format of explanation is provided for various SE tasks?**
- **RQ<sub>3</sub>: How well do XAI techniques perform in supporting various SE tasks?**
  - **RQ<sub>3a</sub>: What baseline techniques are used to evaluate XAI4SE approaches?**
  - **RQ<sub>3b</sub>: What benchmarks are used for these comparisons?**
  - **RQ<sub>3c</sub>: What evaluation metrics are employed to measure XAI4SE approaches?**

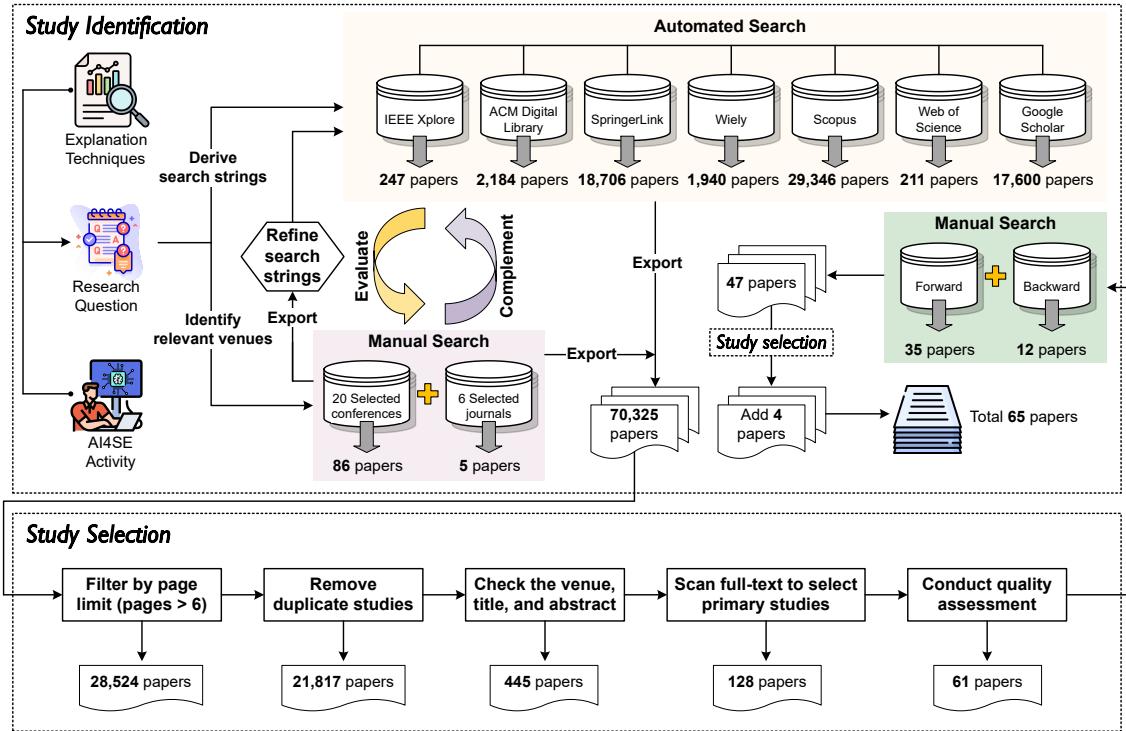


Fig. 2. Study identification and selection process.

### 3.2 Search Strategy

As shown in Fig. 2, following the standardized practice within the field of SE [128], our first step involves identifying primary studies to enhance our ability to address the formulated RQs effectively. Given that the DL revolution – triggered by AlexNet [42] in 2012 – has transformed AI research and became the catalyst for the ML/DL boom in all fields including SE, we chose a 12-year period of January 1st, 2012, to December 31st, 2023, to collect the literature related to XAI4SE. Next, we identified the top peer-review and influential conference and journal venues in the domains of SE and Programming Languages (PL), as outlined in Table 2. In total, we included 15 conferences (ICSE, ASE, ESEC/FSE, ICSME, ICPC, RE, ESEM, ISSTA, MSR, SANER, ISSRE, COMPSAC, QRS, OOPSLA, PLDI) and six journals (TSE, TOSEM, EMSE, JSS, IST, ASEJ). We chose to include PL venues in our study given the frequent overlap of SE and PL research. Furthermore, we also include five top conferences (AAAI, ICML, ICLR, NeurIPS, IJCAI) that centered on machine learning (ML) and deep learning (DL) as these conferences might feature papers applying ML and DL techniques to SE tasks.

Apart from manually searching primary studies from top-tier venues, we also retrieved relevant papers from five popular digital libraries, including IEEE Xplore<sup>1</sup>, ACM Digital Library<sup>2</sup>, SpringerLink<sup>3</sup>, Wiley<sup>4</sup>, and Scopus<sup>5</sup>, and two of

<sup>1</sup><https://ieeexplore.ieee.org>

<sup>2</sup><https://dl.acm.org>

<sup>3</sup><https://link.springer.com>

<sup>4</sup><https://onlinelibrary.wiley.com>

<sup>5</sup><https://www.scopus.com>

Table 2. Publication Venues for Manual Search

Venue	Acronym	Full name
Conference	ICSE	ACM/IEEE International Conference on Software Engineering
	ASE	IEEE/ACM International Conference Automated Software Engineering
	ESEC/FSE	ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering
	ICSME	IEEE International Conference on Software Maintenance and Evolution
	ICPC	IEEE International Conference on Program Comprehension
	RE	IEEE International Conference on Requirements Engineering
	ESEM	ACM/IEEE International Symposium on Empirical Software Engineering and Measurement
	ISSTA	ACM SIGSOFT International Symposium on Software Testing and Analysis
	MSR	IEEE Working Conference on Mining Software Repositories
	SANER	IEEE International Conference on Software Analysis, Evolution and Reengineering
	ISSRE	IEEE International Symposium on Software Reliability
	COMPSAC	IEEE International Computer Software and Applications Conference
	QRS	IEEE International Conference on Software Quality, Reliability and Security
	OOPSLA	ACM SIGPLAN International Conference on Object-oriented Programming, Systems, Languages, and Applications
	PLDI	ACM SIGPLAN Conference on Programming Language Design and Implementation
	AAAI	AAAI Conference on Artificial Intelligence
	ICML	International Conference on Machine Learning
	ICLR	International Conference on Learning Representations
	NeurIPS	Annual Conference on Neural Information Processing Systems
	IJCAI	International Joint Conference on Artificial Intelligence
Journal	TSE	IEEE Transactions on Software Engineering
	TOSEM	ACM Transactions on Software Engineering and Methodology
	EMSE	Empirical Software Engineering
	JSS	Journal of Systems and Software
	IST	Information and Software Technology
	ASEJ	Automated Software Engineering

Table 3. Search Keywords

Group	Keywords
1	"Machine Learn <sup>*</sup> " OR "Deep Learning" OR "Neural Network?" OR "Reinforcement Learning"
2	"Explainable" OR "Interpretable" OR "Explainability" OR "Interpretability"
3	"Software Engineering" OR "Software Analytics" OR "Software Mainten <sup>*</sup> " OR "Software Evolution" OR "Software Test <sup>*</sup> " OR "Software Requirement?" OR "Software Develop <sup>*</sup> " OR "Project Management" OR "Software Design <sup>*</sup> " OR "Dependability" OR "Security" OR "Reliability"
4	"Code Representation" OR "Code Generation" OR "Code Comment Generation" OR "Code Search" OR "Code Localization" OR "Code Completion" OR "Code Summarization" OR "Method Name Generation" OR "Bug" OR "Fault" OR "Vulnerability" OR "Defect" OR "Test Case" OR "Program Analysis" OR "Program Repair" OR "Clone Detection" OR "Code Smell" OR "SATD Detection" OR "Compile" OR "Code Review" OR "Code Classification" OR "Code Change" OR "Incident Detection" OR "Effort Cost Prediction" OR "GitHub" OR "StackOverflow" OR "Developer"

\* is a wildcard used to match zero or more characters.

? is another wildcard used to match a single character.

the most popular research citation engines, Web of Science<sup>6</sup> and Google Scholar<sup>7</sup>, based on the search string (listed in

<sup>6</sup><https://www.webofscience.com>

<sup>7</sup><https://scholar.google.com>

Table 4. Summary of the Process of Study Search and Selection

Data Source	# Studies
IEEE Xplore	247
ACM Digital Library	2,184
SpringerLink	18,706
Wiely	1,940
Scopus	29,346
Web of Science	211
Google Scholar	17,600
Merge	70,325
Filtering studies less than 6 pages	28,524
Removing duplicated studies	21,817
Excluding primary studies based on venue, title, and abstract	445
Excluding primary studies based on full text	128
After Quality Assessment	61
After Forward & Backward Snowballing	108
Final	63

Table 3) assembled from a group of topic-related keywords summarized from manually collected papers. As shown in Table 4, we collected a total of 70,325 relevant studies with the automatic search from these seven electronic databases.

### 3.3 Study Selection

3.3.1 *Inclusion and Exclusion Criteria.* After paper collection, we performed a relevance assessment according to the following inclusion and exclusion criteria:

- ✓ *The paper must be written in English.*
- ✓ *The paper must be a peer-reviewed full research paper published in a conference proceeding or a journal.*
- ✓ *The paper must have an accessible full text.*
- ✓ *The paper must adopt ML/DL techniques to address SE problems.*
- ✗ *The paper has less than 6 pages.*
- ✗ *Books, keynote records, non-published manuscripts, and grey literature are dropped.*
- ✗ *The paper is a literature review or survey.*
- ✗ *The paper is not a conference paper that has been extended as a journal paper.*
- ✗ *The paper uses SE approaches to contribute to ML/DL systems.*
- ✗ *The studies that do not apply XAI techniques on SE tasks are ruled out.*
- ✗ *The studies where explainability is discussed as an idea or part of the future work of a study are excluded.*

In particular, by literature filtering and deduplication (exclusion criteria 1), the total number of included papers was reduced to 21,817. After the first two authors manually examined the venue, title, and abstracts of the papers, the total number of included papers declined substantially to 445. Any ambiguous papers would be forwarded to the fourth and 11th authors who were experienced in the fields of SE and XAI research to conduct a secondary review. In addition, books, keynote records, non-published manuscripts, grey literature, SLRs/surveys, and conference versions of extended papers were also discarded in this phase (exclusion criteria 2-4). The SE4AI papers [2], which used SE approaches to contribute to ML/DL systems were also not considered (exclusion criteria 5) because our SLR focused exclusively on the Manuscript submitted to ACM

Table 5. Checklist of Quality Assessment Criteria for Explainability Studies in AI4SE

No.	Quality Assessment Criteria
QAC <sub>1</sub>	Is the impact of the proposed approach (or empirical/case study) on the AI4SE community clearly stated?
QAC <sub>2</sub>	Are the contributions of the study clearly claimed?
QAC <sub>3</sub>	Does the study provide a clear description of the workflow and implementation of the proposed approach?
QAC <sub>4</sub>	Are the experiment details, including datasets, baselines, and evaluation metrics, clearly described?
QAC <sub>5</sub>	Do the findings drawn from the experiments strongly substantiate the arguments presented in the study?

Table 6. Extracted Data Items and Related Research Questions

RQ	Data Item
RQ <sub>1</sub>	The SE task that an XAI4SE approach tries to solve
RQ <sub>1</sub>	The SE activity in which each SE task belongs
RQ <sub>1</sub>	Publication type of each primary study (i.e., new technique, empirical study, or case study)
RQ <sub>2</sub>	XAI technique employed by each study
RQ <sub>2</sub>	Explanation format
RQ <sub>3</sub>	The adopted baseline approaches
RQ <sub>3</sub>	Benchmark dataset name
RQ <sub>3</sub>	Presence/absence of replication package
RQ <sub>3</sub>	What metrics are used to evaluate the XAI techniques

explainability of AI4SE models. Furthermore, we ruled out studies that did not apply XAI techniques on SE tasks, or just discussed explainability as an idea or future work (exclusion criteria 6,7). In the fourth phase, we reviewed the full texts of the papers (inclusion criteria 3), identifying 128 primary studies directly relevant to our research topic.

**3.3.2 Quality Assessment.** To prevent biases introduced by low-quality studies, we formulated five **Quality Assessment Criteria (QAC)**, given in Table 5, to evaluate the 128 included studies. The quality assessment process was piloted by the first and second authors, involving 30 randomly selected primary studies. We adopted pairwise inter-rater reliability with Cohen's Kappa statistic [10] to measure the consistency of the markings. For any case that they did not reach a consensus after open discussions, the fourth and 11th authors (domain experts experienced in SE and XAI) were consulted as tie-breakers. Within two iterations, the Cohen's Kappa coefficient was successfully raised from *moderate* (0.58) to *almost perfect agreement* (0.84). Then, an assessment was performed for the remaining 98 primary studies. After quality assessment, a final set of 61 high-quality papers was reserved.

**3.3.3 Forward and Backward Snowballing.** To avoid omitting any possibly relevant work during our manual and automated search process, we also performed lightweight backward and forward snowballing [112], i.e., basically examining the research referenced in each of our selected primary studies, as well as the publications that subsequently referred to these studies, on the references and the citations of 61 high-quality papers. As a supplement, we gathered 47 more papers, and conducted the complete study selection process again, including filtering, deduplication, and quality assessment, and obtained two additional papers.

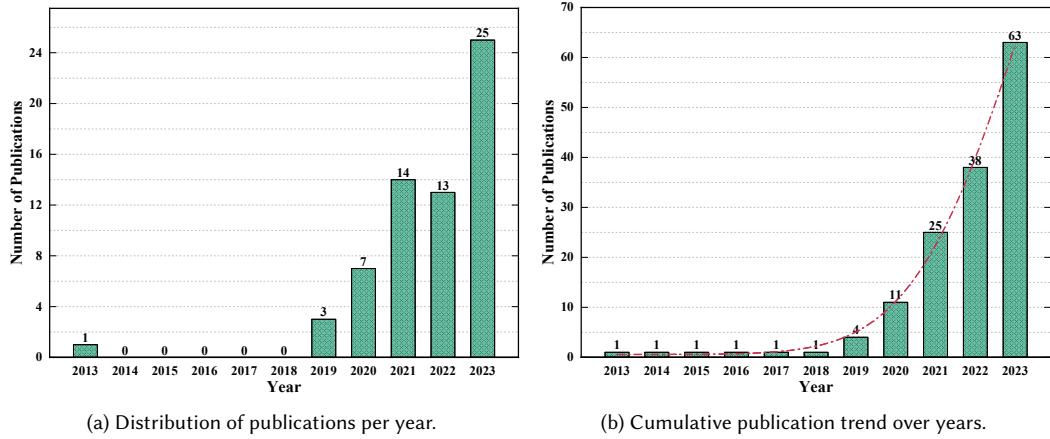


Fig. 3. Publication statistics over years.

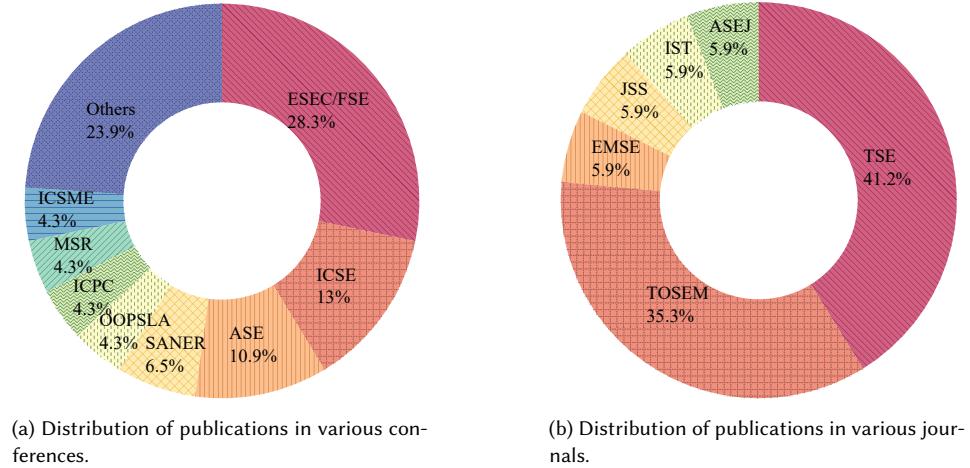


Fig. 4. Distribution of selected papers in different publication venues.

### 3.4 Data Extraction and Analysis

Based on the collected 63 primary studies, we extracted the essential data items used to answer three main RQs. In Table 6, we outline the details information extracted and gathered from 63 primary studies. The column labeled “Data Item” enumerates the relevant data items extracted from each primary study, while the column “RQ” specifies the corresponding research question. In order to mitigate errors during data extraction, the first two authors working together on extracting these data items from the primary studies. Then, the fifth author verified the extracted data results.

Fig. 3a presents the distribution of selected primary studies in each year. The first XAI4SE study we found was published in 2013 [84]. After that, there was a 5-year research gap, ranging from 2014 to 2018. The enthusiasm for investigating the explainability on AI4SE models has steadily risen since 2019, and reaches its peak in 2023, comprising Manuscript submitted to ACM

Table 7. SE Task Taxonomy

SE Activity	SE Task	# Papers	References
Software Development	Code Understanding	6	[9, 75, 79, 99, 103, 117]
	Program Synthesis	3	[12, 21, 67]
	Code Summarization	3	[30, 39, 71]
	Code Search	1	[100]
Software Testing	Test Case-Related	4	[1, 41, 91, 124]
	Debugging	3	[8, 31, 43]
	Vulnerability Detection	11	[26, 36, 47, 52, 68, 90, 93, 129, 130, 135, 137]
	Bug/Fault Localization	2	[48, 110]
Software Maintenance	Program Repair	1	[58]
	Malware/Anomaly Detection	4	[46, 51, 113, 132]
	Bug/Defect Prediction	12	[17, 28, 29, 40, 44, 57, 63, 74, 76, 107, 116, 133]
	OSS Sustainability Prediction	1	[114]
	Incident Prediction	1	[131]
	Root Cause Analysis	1	[15]
	Code Review	1	[118]
	Code Smell Detection	3	[77, 102, 122]
	Bug Report-Related	2	[14, 84]
Software Management	Information Search on Q&A Sites	1	[50]
	Configuration Extrapolation	1	[16]
	Effort/Cost Estimation	1	[27]
	Developer Recommendation	1	[115]

39.7% of the total publications. Fig. 3b illustrates the cumulative publication trend over years. It is observable that the slope of the curve fitting the distribution experiences a significant increase between 2019 and 2023. This pronounced upward trend indicates a burgeoning research interest in the field of XAI4SE.

We also analyzed the publication trend of primary studies in selected conferences and journal venues, respectively. As shown in Fig. 4a, ESEC/FSE stands out as the predominant conference venues favored by XAI4SE studies, with a contribution of 28.3% of the total. Other venues making noteworthy contributions include ICSE (13%), ASE (10.9%), and SANER (6.5%). Fig. 4b shows the distribution of primary papers published in different journal venues. It can be seen that 76.5% of relevant papers were published in TSE and TOSEM, which indicates a booming trend of XAI4SE research in top-tier SE journals in the past few years.

#### 4 RQ<sub>1</sub>: 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. Out of the 63 primary studies we analyzed for this SLR, we identified 21 separate SE tasks where an XAI technique had been applied. These tasks span across the four main phases of the **Software Development Life Cycle (SDLC)** [82], including software development, software testing, software maintenance, and software management. The full taxonomy is displayed in Table 7, which associates the relevant primary study paired with the SE task & activity it belongs to.

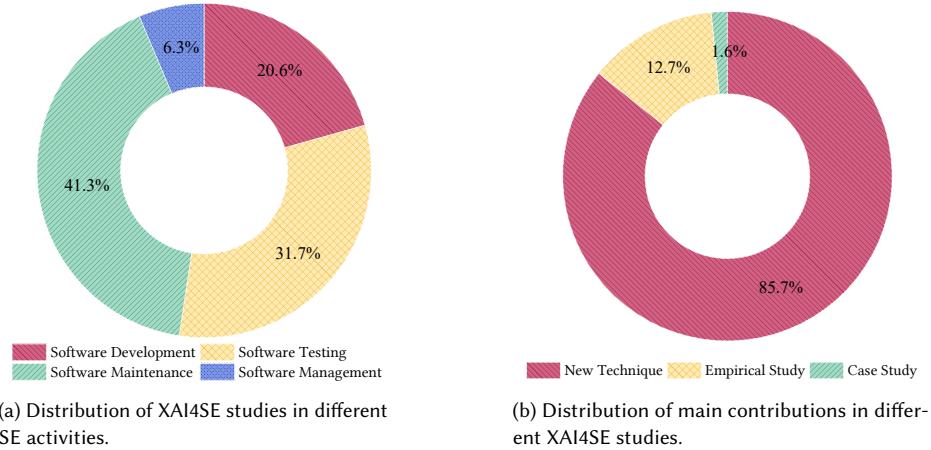


Fig. 5. Distribution of XAI4SE studies across different SE activities and contribution types.

#### 4.1 Exploratory Data Analysis

Fig. 5a describes the distribution of 63 primary studies in four SE activities. It is noteworthy that the highest number of studies is observed in software maintenance, comprising 41.3% of the total research volume. Following that, 31.7% of studies were dedicated to software testing, and 20.6% of studies focused on solving SE tasks in software development. This distribution underscores the vital focus on development and maintenance tasks. By contrast, software management only accounts for a marginal proportion (6.3%) of the research 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 paper, and then grouped them into three categories, i.e., *New Technique*, *Empirical Study*, and *Case Study*. As shown in Fig. 5b, 85.7% of the primary studies concentrated on proposing novel explainable solutions to improve the transparency of black-box AI models in various SE tasks. Another 12.7% of the relevant research focused on conducting empirical studies to investigate the advantages and disadvantages of state-of-the-art XAI4SE approaches from different perspectives, e.g., analyzing the performance differences between off-the-shelf XAI tools over certain SE tasks/models such as defect prediction [40]. The remaining one (1.6%) primary study performed a case study to explain the performance of ChatGPT [69] under distinct prompt engineering approaches [79].

Fig. 6 displays a visual breakdown of different SE tasks we found within 63 primary studies across a 12-year period. Early SE tasks to explore the explainability were those of *Bug/Defect Prediction*, *Program Repair*, *Code Smell Detection*, and *Bug Report-Related Automation*. It was not until 2021 that the diversity of target SE tasks experienced a significant increase, including *Code Understanding*, *Debugging*, and *Malware/Anomaly Detection*, and so on. It is noteworthy that there are three SE tasks that have consistently maintained high activity levels over the years: *Bug/Defect Prediction*, *Vulnerability Detection*, and *Code Understanding*. The most popular of the three is **Bug/Defect Prediction**, contributing  $\approx 19\%$  of the papers we collected. We suspect that the main reason contributing to the popularity of XAI in bug/defect prediction is that, expert knowledge-based feature engineering is easier to understand than complex high-dimensional features. Defect prediction model is often built upon a set of hand-crafted software metrics. Thus, developers can easily obtain explainable results through leveraging intrinsically interpretable models (e.g., decision tree) or off-the-shelf post-hoc explainers (e.g., LIME [78]).

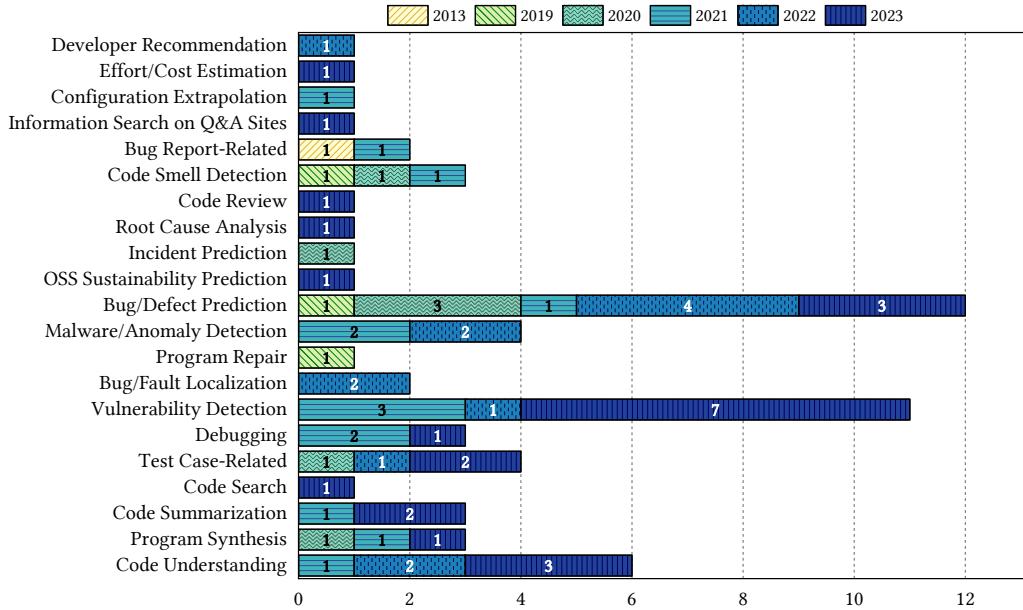


Fig. 6. Papers published per year according to SE task.

#### 4.2 How XAI Were Used in Specific SE Tasks?

In this subsection, we delved into the progress of various SE tasks 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 in advancing practices for explainable AI4SE solutions.

**4.2.1 SE Tasks in Software Development.** There are wide-ranging applications of XAI techniques in software development, encompassing code understanding, program synthesis, code summarization, and code search.

**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, such as such as *Variable Misuse Detection* [75], *Method Name Prediction* [103], and *Vulnerability Detection* [117]. 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 [37]) or unlabeled code corpus (e.g., CodeXGlue [53]). This training process produces code embeddings with rich contexts and semantics. Yang et al. [117] 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. [99] 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.

**Code Summarization.** Code summarization, also known as *code comment generation*, is a *code-to-text* task that attempts to automatically generate textual descriptions directly from source code. A promising solution is **Neural Comment Generation (NCG)**, which employs sequence-to-sequence neural networks such as **Neural Machine Translation (NMT)** [7, 94] to auto-summarize target source code into short natural language text. Despite their effectiveness, state-of-the-art NCG approaches are still far from being usable in practice because developers are hard to understand and trust such end-to-end summarization without any explanation. To this end, Geng et al. [30] designed two types of explanation strategies applicable to different application scenarios to identify the corresponding code parts used to generate summarization. The black-box explanation strategy aimed to localize the code segment that sensitive to program mutations, while the white-box explanation strategy focused on inspecting the attention score of each code token. Jiang et al. [39] proposed CCLink, a model-independent framework which aimed to find the code segments that contribute to the generation of key information in the auto-generated comments. CCLink generated a series of code mutants from key phrases in the auto-generated comment, and tailored data mining algorithms to construct the links between code and its auto-generated comment. This in turn allowed CCLink to visualize links as the comment explanations to developers.

**Code Search.** Code search is the *text-to-code* task of retrieving source code that meets users' natural language queries from a large codebase. Deep learning-based approaches, which maps query/code to feature embeddings and measures their similarity, has become the leading paradigm for code search. However, due to the lack of relevant explanatory information, it is time-consuming and labor-intensive for developers to understand the ranked list of candidate code snippets. Wang et al. [100] proposed an explainable code search tool, namely XCoS, to bridge the knowledge gap between the query and candidate code snippets. Based on the background knowledge graph extracted from Wikidata and Wikipedia, XCoS provided conceptual association paths, relevant descriptions, and additional suggestions, as explanations. Furthermore, it designed an interactive User Interface (UI) which organized explanatory information in the form of trees to help developers intuitively understand the rationale behind the search results.

4.2.2 *SE Tasks in Software Testing.* The applications of XAI in software testing encompass tasks such as test case-related, debugging, vulnerability detection, and bug/fault localization.

**Test Case-Related.** The design and implementation of test cases occupy most of the software testing cycle because their quality directly affects the quality of software testing. In recent years, the combination of intelligent technology and test scenarios, such as test case generation [124], test case recommendation [41], and test case diversity analysis [1], has received widespread attention. Yu et al. [124] improved the DL-based automated assertion generation approach by integrating the **Information Retrieval (IR)**-based assertion retrieval technique and the retrieved-assertion adaptation technique. The assertion retrieval using IR also yields the corresponding focal-test. In this context, focal-test refers to a pair consisting of a test method without an assertion and its focal method. Developers can then use this focal-test as a valuable reference during assertion inspection. To make full use of historical test cases to improve test efficiency, Ke et al. [41] proposed an explainable test case recommendation approach based on the knowledge graph. Once a historical test case is predicted to be prone to revealing defects by the classifier, the corresponding knowledge chain in the software test knowledge graph will be returned as auxiliary information to help testers understand the reason for the recommendation.

**Debugging.** ML/DL-based models for SE tasks, similar to traditional software, suffer from errors that result in unexpected behavior or incorrect functionality. Due to the black-box nature of complex AI models, the debugging process designed for traditional software, which involves reviewing the code, tracing abnormal execution flows, and isolating the root

cause of the problem, may not be applicable to ML/DL-based SE models. Motivated by the idea that diagnostic features could be potentially useful, Cito et al. [8] presented a model-agnostic rule induction technique to explain when a code model performed poorly. The misprediction diagnosis model aimed to learn a set of rules that collectively cover a large portion of the model's mispredictions, each of which correlates strongly with model mispredictions. The evaluation results showed that these learned rules are both accurate and simple.

**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 software engineering 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. [47] leveraged GNNExplainer [123] 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 [26], root cause [90], similar vulnerability reports [68], and so on. For instance, Zhou et al. [135] 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.

**Bug/Fault Localization.** Bug localization refers to the process of pinpointing the exact location in the codebase where the bug originates based on bug reports or issue descriptions provided by users or testers. From the perspective of enhancing the effectiveness of bug localization, Widyasari et al. [110] formulated the localization task as a binary classification problem, i.e., predicting whether a test case will fail or pass. They applied TreeSHAP [54], a local model-agnostic explanation technique, to identify which parts of code are important in each failed test case. From the perspective of providing explanations for localized bugs, Li et al. [48] respectively designed the global and local explanation strategies to explain the model predictions. The global explanation strategy leveraged decision trees as the surrogate models to infer the decision paths leading to only faulty or normal failure units, while the local explanation strategy compared the incoming failure with each historical failure to explain how the model diagnosed a given failure.

**4.2.3 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, code smells, and bug report-related.

**Malware/Anomaly Detection.** As the emerging collaborative software development modes of open source have become increasingly popular, the overall security risk trend, such as malicious applications (malware) and commits, in complex and intertwined software supply relationships increases significantly. Wu et al. [113] proposed an explainable ML-based approach, named XMal, to not only predict whether an app is malware, but also unveil the rationale behind its prediction. For this purpose, XMal built a semantic database based on malware key features and functional descriptions in Android developer documentation<sup>8</sup>, and leveraged the mapping relation between the malicious behaviors and their corresponding semantics to generate descriptions for malware.

---

<sup>8</sup><https://developer.android.google.cn/docs>

**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 feature metrics extracted from a software product to construct the classification model. These software metrics are easy for researchers and participants to understand and use, and give a vivid description of the software's running state. Although non-linear algorithms like neural networks have achieved promising prediction performance compared to traditional tree or rule-based models, they are hard to provide both accurate and explainable prediction results. Yang et al. [116] proposed a weighted association rule based on the contribution degree of features to optimize the process of rule generation, ranking, pruning, and prediction. In terms of explainability, such a weighted ensemble model incorporating multiple rules helped to improve the quality and multiplicity of the rule set. In addition, some studies directly adopt source code (e.g., code tokens) as meaningful semantic units for defect prediction. Wattanakriengkrai et al. [107] formalized defect explanation as a line-level prediction task, and used the model-agnostic technique LIME [78] to identify risky code tokens in predicted defective files. A comparative evaluation across 32 releases of nine software systems showed that leveraging a model-agnostic technique can effectively identify and rank defective lines that contain common defects while requiring a smaller amount of inspection effort and a manageable computation cost.

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. [133] 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. Similarly, Pornprasit et al. [74] leveraged the crossover and mutation operations to randomly generate synthetic neighbors around the to-be-explained instance based on the actual characteristics of defect-introducing commits, and built a local rule-based regression model to explain the interactions between defect-introducing features.

**Code Smell Detection.** Due to the delivery date pressure or developer oversight, code smells may be introduced in software development and evolution, thereby reducing the understandability and maintainability of source code. One of the common code smells is **Technical Debt (TD)**, which reflects the trade-off software engineers make between short-term benefits and long-term stability. TD is accumulated when developers make wrong or unhelpful technical decisions, either intentionally or unintentionally, during software development. Despite the effectiveness of DL-based code smell detection approaches in automatically building the complex mapping between code features and predictions, they cannot provide the rationale for the prediction results. To understand the basis of the CNN-based detection model's decisions, Ren et al. [77] proposed a backtracking-based approach, which exploited the computational structure of CNN to derive key phrases and patterns from input comments.

**Bug Report-Related.** A Bug report is one of the important crucial SE documentation for software maintenance. High-quality bug reports can effectively reduce the cost of fixing buggy programs. For example, instead of using other black-box models, Shihab et al. [84] employed an explainable ML model (i.e., decision tree) to understand which attributes affected whether or not a bug would be re-opened. To analyze the impact of test-related factors to DL-based bug report prediction models, Ding et al. [14] applied SHAP to compute the importance of tell smell features.

**Others.** Apart from the above tasks, some studies leveraged XAI techniques on certain special SE tasks. For example, the model-agnostic explanation technique LIME [78] was used to explain the prediction results of XGBoost-based SE tasks such as OSS sustainability prediction [114] and incident prediction [131]. In order to mine the formatting style of

each analyzed Git repository, Markovtsev et al. [58] proposed a decision tree-based explainable model to express the found format patterns with compact human-readable rules.

**4.2.4 SE Tasks in Software Management.** There are only four literature involving the utilization of XAI in software management, involving the following main SE tasks, i.e., information search on Q&A sites, configuration extrapolation, effort/cost estimation, and developer recommendation, leaving ample space for further exploration.

**Information Search on Q&A Sites.** As one of the most popular and widely-used technical Q&A sites in SE communities, **Stack Overflow (SO)** plays an increasingly important role in software development. When facing software programming problems such as implementing specific functionalities or handling errors in code, developers often turn to SO for help. To provide both accurate and explainable retrieval results, Liu et al. [50] proposed KGXQR, a knowledge graph-based question retrieval approach for programming tasks. KGXQR constructed a software development-related concept knowledge graph to find the desired questions, and further generated explanations based on the association paths between the concepts involved in the query and the SO questions to bridge the knowledge gap.

**Configuration Extrapolation.** Configuration management is a process in systems engineering for establishing consistency of a product's attributes throughout its life. The increasing configurability of modern software also puts a burden on users to tune these configurations for their target hardware and workloads. To configure software applications efficiently, ML models have been applied to model the complex relationships between configuration parameters and performance. To understand the underlying factors that caused the low performance, Ding et al. [16] used an inherently interpretable linear regression model [23] to find the configuration with the best predicted performance. They provided interactive visualization charts (e.g., radar charts, bar charts) to explain the relationships between the application-level configuration parameters and ultimate performance.

**Effort/Cost Estimation.** Effort/Cost estimation is the process of predicting 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. [27] 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.

**Developer Recommendation.** Collaboration efficiency is of paramount importance for software development. Although a lot of efforts in recommending suitable developers have been made in both research and practice in recent years, such approaches often suffer from low performance due to the difficulty of learning the developer's expertise, willingness, relevance as well as the sparsity of explicit developer-task interactions. Xie et al. [115] proposed a multi-relationship embedded approach named DevRec, in which they explicitly encoded the collaboration relationship, interaction relationship, and similarity relationship into the representation learnings of developers and tasks. DevRec also visualized the high-order connectivity and attentive embedding propagation in the recommendation subgraphs to explain why a task was recommended (or assigned) to the developer.

◀ ▶ RQ<sub>1</sub> - Summary ◀

- We summarized a total of 63 primary studies into 21 SE tasks, which cover four major activities within SDLC, including software development, software testing, software maintenance, and software management. Subsequently, we delved into the progress of various SE tasks that applied XAI techniques.
- In the span of the 12-year period, research on explainability of AI4SE models has experienced a notable shift in preference. Early XAI4SE studies predominantly concentrated on ***bug/defect prediction, program repair***, etc. It was not until 2021 that the diversity of target SE tasks witnessed a significant increase, including ***code understanding, debugging, and malware/anomaly detection***.
- We found that the explainability of learning-based ***vulnerability detection*** and ***defect prediction*** models are the most widely explored (with 11 and 12 papers, respectively). The least mentioned SE activity, software management, was involved in only four papers.

## 5 RQ<sub>2</sub>: HOW XAI TECHNIQUES ARE USED TO SUPPORT SE TASKS?

In Section 4, 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.

### 5.1 RQ<sub>2a</sub>: 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 taxonomies 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 [86]. In addition, these taxonomies are mostly developed for general purpose or specific downstream applications such as healthcare [59] and finance [121], 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, according to their methodologies of implementation, most XAI techniques studied in this review can be categorized into five groups: *Interpretable Model (IM)* ( $\approx 25\%$ ), *Feature Attribution (FA)* ( $\approx 33\%$ ), *Attention Mechanism (AM)* ( $\approx 13\%$ ), *Domain Knowledge (DK)* ( $\approx 21\%$ ), and *Example Subset (ES)* ( $\approx 6\%$ ). Fig. 7 illustrates the various types of XAI techniques that we extracted from our selected studies.

**Interpretable Model (IM).** The easiest way to achieve explainability is to construct interpretable models, such as Decision Tree (DT), Logistic Regression (LR), Rule-based Models (RM), and Naive Bayes (NB). These models have built-in explainability by nature. For instance, in DT and RM, the association between features and target is modelled linearly or monotonically, i.e., an increase in the feature value either always leads to an increase or always to a decrease in the target outcome. Such linearity or monotonicity is useful for the interpretation of a model because it makes it easier to understand a relationship. On the other hand, NB and Bayesian networks formulate acyclic graphs to provide conditional probabilistic outcomes presenting the relationship of variables. As the most representative model, decision tree predicts the value of a target variable by learning simple *if-then-else* rules inferred from the data features. The

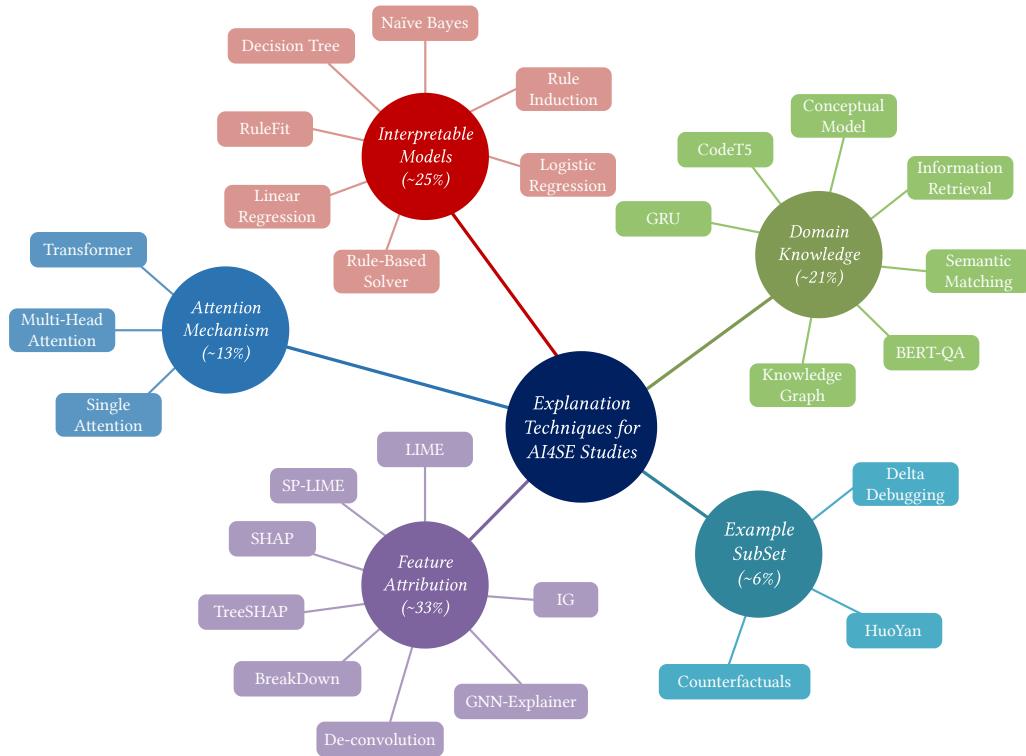


Fig. 7. XAI techniques taxonomy &amp; distribution.

tree structure is ideal for capturing interactions between features in the data, and also has a natural visualization of a decision making process. Taking Fig. 8 as an example, each node in a decision tree 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 [84]. In addition, interpretable models 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, e.g., deep neural networks, SVMs, or ensemble models. Examples include vulnerability detection [137], defect prediction [74], and program repair [58].

**Feature Attribution (FA).** Feature attribution-based explanations aim to measure the relevance (i.e., attribution score) of each input feature to a model's prediction by ranking the explanatory score. The score could range from a positive value that shows its contribution to the model's prediction, to a zero that would mean the feature has no contribution, to a negative value which means that removing that feature would increase the probability of the predicted class. They can be broadly divided into (1) perturbation-based approaches that make feature perturbations while analyzing prediction change, e.g., LIME [78], GNNExplainer [123], and SHAP [55], (2) gradient-based approaches that propagate importance signals backward through all neurons of the network, e.g., Integrated Gradients (IG) [92] and Grad-CAM [83], and (3)

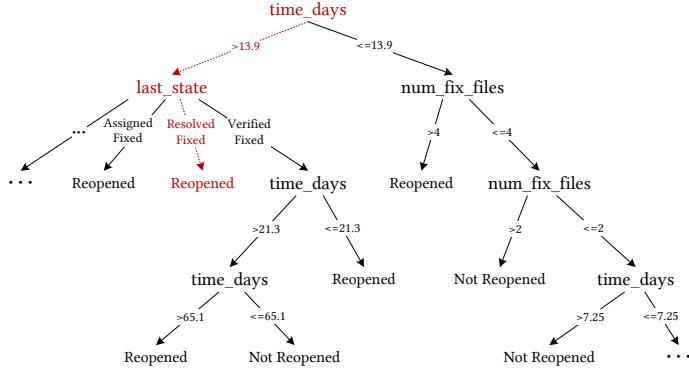


Fig. 8. Sample decision tree used for explainable re-opened bug prediction.

decomposition-based approaches that break down the relevance score into linear contributions from the input, e.g. Layer-wise Relevance Propagation (LRP) [4], Deep Taylor Decomposition (DTD) [62], and DeepLIFT [85].

Among them, feature perturbations are perhaps the most popular approaches in XAI4SE studies. These perturbations can be performed in high-dimensional latent space by masking [47, 130] or modifying [74, 114] certain features of input instances. The basic idea is to remove the minimum set of inputs to change the model prediction. An early representative perturbation-based approach is **Local Interpretable Model-agnostic Explanations (LIME)** [78]. 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 surrogate model (e.g., decision tree, linear regression) to produce an explanation. LIME has been successfully applied to various SE tasks, such as defect prediction [44, 107, 133], OSS Sustainability Prediction [114], and test case generation [1]. **SHapley Additive exPlanations (SHAP)** [55], which stems from game theory, is another popular XAI technique based on perturbation-based feature attribution. It assigns each feature an 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. [110] applied a tree ensemble model-specific variant, TreeSHAP [54], to identify which code statements are important in each failed test case.

**Attention Mechanism (AM).** Attention mechanism is often viewed as a way to attend to the most relevant part of inputs. Intuitively, attention may capture meaningful correlations between intermediate states of input that can explain the model's predictions. The most significant advantage of attention mechanism lies in that it can be easily integrated into any deep neural network (e.g., CNN [71], Bi-LSTM [102], GRU [30], GNN [46], Transformer [26]), and improve the performance of the original task as well as providing explainability. Many approaches try to explain models based on the attention weights or by analyzing the knowledge encoded in the attention. For example, Li et al. [46] employed an attention-based GNN model, named GAT [97], to weigh the importance of neighboring code graph nodes in runtime exception detection. These computed weights were then used to visualize the importance of various edges to the detected runtime exception. Similarly, Wang et al. [102] visualized the feature weight of each word with the help of single- and multi-head attention mechanism to offer word- and phase-level explainability for why a comment is classified as SATD.

**Domain Knowledge (DK).** So far, we have encountered several XAI techniques to explain black-box models. However, there are some limitations regarding them in practice. On the one hand, due to the weak capability of processing complex data, intrinsically interpretable models are prone to getting trapped into the trade-off dilemma between performance and explainability, i.e., sacrificing predictive accuracy in exchange for explainability. On the other hand, important regions highlighted by feature attribution or attention mechanism are not necessarily user-friendly in terms of explainability. For example, the importance of a single token in a code snippet may not convey a sufficiently meaningful explanation. To address these challenges, some works take special steps to incorporate additional knowledge from experts into their explanations. Concretely, we identify two incipient trends in the application of domain knowledge: (1) designing one or more auxiliary tasks related to the main task to provide additional insights regarding the input data, and (2) 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 [26], identify key aspects [90], generate vulnerability descriptions [57, 129], search for similar issues [68], etc. To assist developers in understanding the return results of neural code search tools, XCoS [100] 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.

**Example Subset (ES).** Apart from the above XAI techniques, another form of explanation exists by explaining model behavior from the perspective of sample subsets, which aims to offer insights into the factors that influenced the model's decision from individual instances or scenarios where the model's prediction changes. One representative work in this area is the counterfactual explanation. It treats the input as the cause of the prediction under the Granger causality, i.e., given an observed input  $x$  and a perturbed  $\hat{x}$  with certain features changed, the prediction  $y$  would change to  $\hat{y}$ . Counterfactual explanation reveals what would have happened based on certain observed input changes. They are often generated to meet up certain needs such as misprediction diagnosis by selecting specific counterfactuals. Such examples can be generated by humans or perturbation techniques like word replacement. Cito et al. [9] 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. In addition, some attempts borrowed certain classical techniques, such as program mutation [88] and **Delta Debugging (DD)** [127], from the field of software testing to search for important code snippets positively contributing to the model predictions. Rabin et al. [75] leveraged DD to simplify a piece of code into the minimal fragment without reversing its 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. By integrating with four state-of-the-art neural architectures across two popular tasks (code summarization and variable misuse detection), they showed that model inputs could often be reduced tremendously (even by 90% or more), and these reductions in no way required the inputs to remain natural, or, depending on the task, in any way valid. Suneja et al. [93] similarly implemented a prediction-preserving input minimization (P2IM) approach to quantify how well the black-box AI model learned the correct vulnerability-related signals.

**5.1.1 Exploratory Data Analysis.** Fig. 9 delineates the application status of different types of explanation approaches in 21 previously summarized SE tasks. Overall, feature attribution is the most prevalent explainability technique in XAI4SE research, followed by interpretable model, domain knowledge, and attention mechanism. Given the flexibility

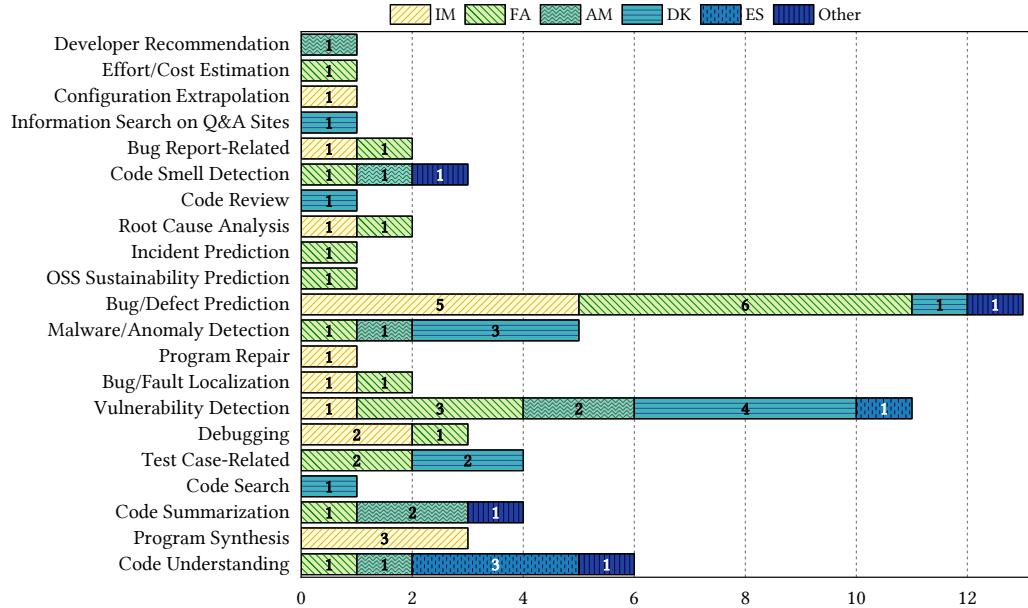


Fig. 9. XAI techniques by the task.

of feature attribution in model adaptation, the popularity of such types of XAI techniques is not surprising. Meanwhile, the prevalence of interpretable model is also expected, 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, domain knowledge-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 extracted the selection rationales for XAI techniques from selected 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 metric-based AI4SE pipelines (e.g., defect prediction [74], OSS sustainability prediction [114], and incident prediction [131]) were derived by using interpretable models, such as decision tree or RuleFit [25], thanks to their strong ability to analyze and extract human-understandable rules from hand-crafted feature metrics.

**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 [36, 47, 130]. For instance, due to the great performance of deconvolution [126] in providing visual explanations for CNN-based applications, Ren et al. [77] leveraged a targeted backtracking technique to extract prominent phrases that contribute most to the decision whether the comment is a SATD or not from the input comment as explanations.

**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 speaking, model users aim to utilize XAI techniques to better understand the output of a deployed model and make an informed decision, while model 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 [107], while others extract human-understandable rules [74], or even natural language descriptions [57] from the defective code that can serve as actionable and reusable patterns or knowledge.

#### ↳ ▶ RQ<sub>2a</sub> - Summary ◀

- Our exploratory data analysis revealed five commonly used XAI techniques, including *Interpretable Model (IM)* ( $\approx 25\%$ ), *Feature Attribution (FA)* ( $\approx 33\%$ ), *Attention Mechanism (AM)* ( $\approx 13\%$ ), *Domain Knowledge (DK)* ( $\approx 21\%$ ), and *Example Subset (ES)* ( $\approx 6\%$ ), that have been used in research on XAI4SE.
- Compared with other XAI techniques, *feature attribution*-based approaches were by far the most popular option in our surveyed studies.
- We summarized the selection strategies for XAI techniques in SE tasks into three main categories: **Task Fitness**, **Model Compatibility**, and **Stakeholder Preference**.

## 5.2 RQ<sub>2b</sub>: What format of explanation is provided for various SE tasks?

Next, in order to gain insights from various explanation formats outputted by XAI4SE approaches, we provide a taxonomy, along with descriptive examples and statistics, as to why some types of explanation formats were used for particular SE tasks. In particular, these explanations can be broadly classified into five major types: *Numeric*, *Text*, *Visualization*, *Source Code*, and *Rule*.

**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 [40, 133] or serve as indicators to guide key feature selection [91]. Popular examples of numerical explanations are LIME [78] and SHAP [55]. Esteves et al. [17] used SHAP values to understand the CK metrics [6] that influenced the defectiveness of the classes. In the same context, Lee et al. [44] employed three widely-used model-agnostic techniques, including LIME, SHAP, and BreakDown [89], to calculate the contribution of each feature for defect models' predictions. Xiao et al. [114] 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. [133] 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. [91] 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 AI models (e.g., NMT [7, 94]) or retrieved from external knowledge

```

14 src/mongo/shell/linenoise.cpp
@@ -2762,7 +2762,17 @@ int linenoiseHistorySave(const char* filename {
2763     /* Save the history in the specified file. On success 0 is returned
2764      * otherwise -1 is returned. */
2765 -     FILE*& fp = fopen(filename, "wt");
2766     if (fp == NULL) {
2767         return -1;
2768     }
2769     for (int j = 0; j < historyLen; ++j) {
2770         if (history[j][0] != '\0') {
2771             fprintf(fp, "%s\n", history[j]);
2772         }
2773     }
2774 -    fclose(fp);
2775     return 0;
2776 }
```

**CVE-2016-6494 Detail**

**Description**

The client in MongoDB uses world-readable permissions on .dbshell history files, which might allow local users to obtain sensitive information by reading these files.

**Weakness Enumeration**

CWE-ID	CWE Name
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor

**VulTeller**

The FILE before VERSION does not verify that use of the permissions on history files allows local guest to read sensitive information by reading these files.

Fig. 10. Vulnerability explanation generated by VulTeller [129], which transfers detected vulnerable code to natural language descriptions conveying the reason for vulnerabilities.

bases (e.g., Stack Overflow<sup>9</sup>, Wikipedia<sup>10</sup>). For example, Mahbub et al. [57] first performed discriminatory pre-training with bug-patch code pairs to equip the pre-trained CodeT5 [104] with the capability of understanding buggy code patterns and generating commit messages accordingly, and then fine-tuned the model to generate explanation from buggy code. Similarly, based on the vulnerable code localized by attention mechanism, Zhang et al. [129] 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. Fig. 10 shows a code snippet of an example (CVE-2016-6494<sup>11</sup>) demonstrated in [129], the proposed approach can capture the semantic information behind the input code snippet and accurately point out the location of the vulnerability, while revealing the potential risk of information leakage as writing sensitive data to the specified file via `fopen()` without checking permissions (at line 2,765).

Another way to obtain textual explanations is by retrieving external knowledge bases. As an example, Wu et al. [113] 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. To bridge the knowledge gap between the query and candidate code snippets, Wang et al. [100] linked both the query and the candidate code snippets to the concepts in the background knowledge graph and generated explanations based on the association paths between these two parts of concepts together with relevant descriptions. Sun et al. [90] extracted vulnerability key aspects (including vulnerability types, root causes, attack vectors, and impacts) from security patches based on pre-trained BERT-based Name Entity Recognition (NER) and Question Answering (QA) models [13], and assembled them into pre-defined templates as explanations. Inspired by similar/homogeneous vulnerabilities that have similar root causes or lead to similar impacts, Ni et al. [68] 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 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 SO and Wikipedia offer structured knowledge models that explicitly store rich factual knowledge. Thus, they

<sup>9</sup><https://stackoverflow.com/>

<sup>10</sup><https://www.wikipedia.org/>

<sup>11</sup><https://nvd.nist.gov/vuln/detail/CVE-2016-6494>

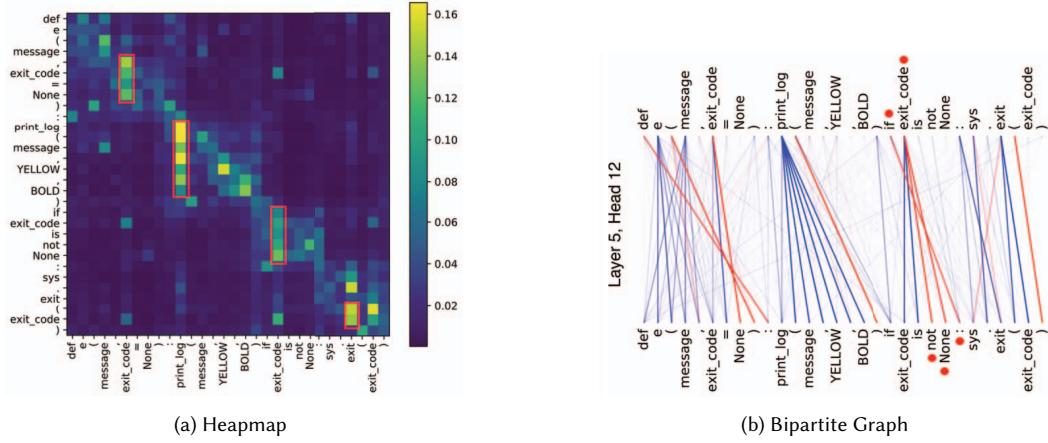


Fig. 11. Visual explanations provided by different techniques. Adapted from [99].

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, which makes the use of attention a particularly attractive option. Humans, in general, can process visual information faster and much easier as compared to other information [64]. Attention was initially introduced to consider correlations between words in a sentence in a parallel fashion and is a primary component in the Transformer architecture [96]. Visualization systems differ in their ability to show relationships at multiple scales, by representing attention in various forms for different models. Fig. 11 displays heatmaps and bipartite graphs, which are commonly used techniques for visualizing attention heads for a single input. These two approaches show distinct visual representations of attention, which are graph and matrix. Liu et al. [52] proposed an attentive multi-encoder network that combined local expert patterns and the global graph feature for vulnerability detection, and produced explanations in the form of a heatmap, highlighting relevant weights. Similarly, Wang et al. [102] visualized the attention weights of the most important words and phrases that have contributed to the model's predictions to explain why a comment is (not) flagged as self-admitted technical debt. To explore why the pre-trained code language models, such as CodeBERT [24] and GraphCodeBERT [34], work and what they indeed capture when applied to SE tasks, Wan et al. [99] further proposed three types of structural analysis techniques, including attention analysis, probing on word embedding, and syntax tree induction.

In addition, some approaches developed interactive User Interface (UI) to provide visual explanations. For example, Jiang et al. [39] 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. Such illustration is light-weighted, which does not incur noticeable latency to the developers. To facilitate examining the candidate code snippets based on related concepts and descriptions to give control back to the user, XCoS [100] leveraged a structured conceptual tree to display the results of query scoping, the

Table 8. The Various Formats of Explanations in Prior Studies

Format	Examples	# Studies	References
Numeric	Importance Score, Similarity	14	[1, 14, 15, 17, 40, 43, 44, 51, 63, 91, 114, 122, 131, 133]
Text	Descriptions, Key Phases/Words	12	[26, 27, 41, 50, 57, 68, 77, 90, 113, 118, 129, 135]
Visualization	Heatmap, Bipartite Graph	9	[16, 39, 46, 52, 99, 100, 102, 115, 132]
Source Code	Tokens, Statements	15	[9, 28, 30, 36, 47, 71, 75, 79, 93, 103, 107, 110, 117, 124, 130]
Rule	Association Rules	13	[8, 12, 21, 29, 31, 48, 58, 67, 74, 76, 84, 116, 137]

identified conceptual association paths. Users can interact by selecting the anchor concept relevant to the query which will update the results of code snippets related to the selected concepts.

**Source Code.** As source code is one of the primary data type that AI4SE models attempt to learn from, it serves as a natural format of explanation. Geng et al. [30] 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. Wang et al. [103] further classified an entire input into two types of features: defining features (i.e., *wheat*) representing the reasons models predict a specific label, and the rest (i.e., *chaff*). They proposed a coarse-to-fine approach, named "*Reduce and Mutate*", to identify *wheat* that code models use for prediction. In contrast to simplification-based techniques that can be applied to any DL architecture, Li et al. [47] employed a GNN-specific explanation framework, GNNExplainer [123], to simplify the target code instance to a minimal subset consisting of crucial statements while retaining the initial model prediction. Along the lines of architecture-specific simplification, Hu et al. [36] investigated six famous GNN explainers to evaluate their explanation performance on vulnerability detectors from three different perspectives, which are effectiveness, stability, and robustness. Their empirical study showed that the explanation results provided by different explainers for vulnerability detection vary significantly, and the performance of all explainers was still not satisfactory.

**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. [137] 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. [74] 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. Then, they derived a positively-correlated rule set having the highest importance score and satisfying the actual feature values of the instance as explanations. Cito et al. [8] proposed a rule induction technique which produced decision lists based on features and mispredicted instances to explain the reasons for mispredictions. Based on the prior knowledge that data often exhibit highly-skewed feature distributions and trained models in many cases perform poorly on the subset with under-represented features, Gesi et al. [31] deduced the misprediction explanation rules only on biased features instead of blindly trying on all features, hence deducing better rules within the same computation time.

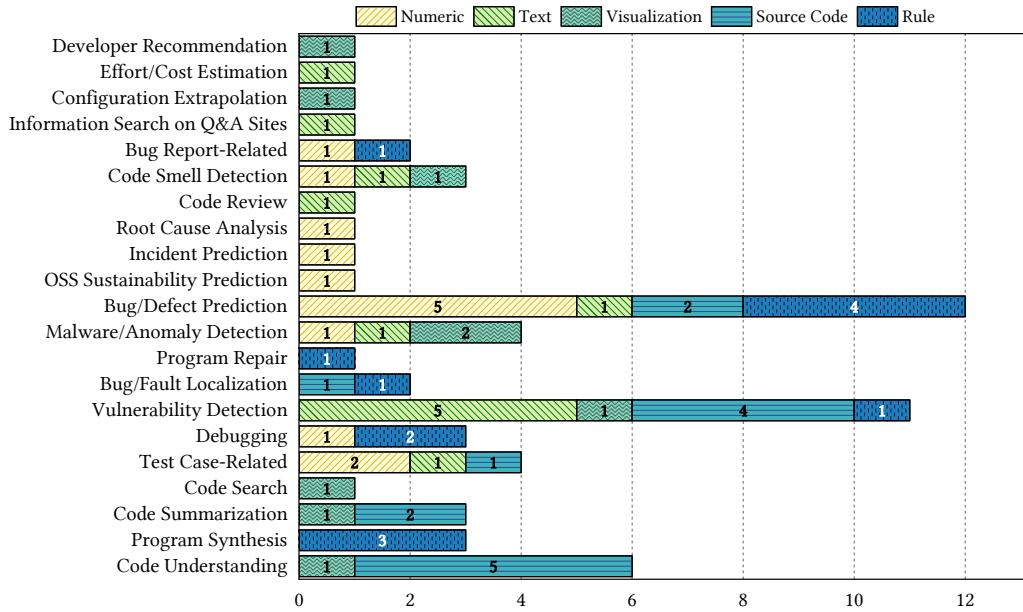


Fig. 12. Explanation formats with SE tasks.

**5.2.1 Exploratory Data Analysis.** Table 8 describes the proportion of various formats of explanations we summarized from 63 primary studies in detail and gives examples. We found that the distribution of different explanation formats was relatively even, and the most common format being employed was *source code* (~24%). The prevalence of source code-based explanation is not surprising, given the rise in the number of publicly available code repositories, a series of AI models [33, 34] have achieved notable advancements in various downstream code-centric SE tasks, such as code generation, understanding, and analysis [98], making it a natural way to obtain human-understandable explanations. In addition, we noticed that visual explanation is less commonly used in the SE community, only accounting for a total of 9 primary studies. We suspect that the main reason contributing to its relative lack of adoption lies in that, although visualization, such as heatmap [52] and bipartite graph [99], can provide a fast and straightforward explanation for practitioners (e.g., a developer, domain expert, or end-user) who are inexperienced in ML/DL [87], it can only convey limited information and requires post-processing for further use.

Fig. 12 provides an overview of the distribution of explanation formats adopted by different SE tasks. We observed that the types of explanation formats 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 [68, 129], types [90, 135]), visualization (e.g., heatmap [52]), source code (e.g., code statements [47, 93]), or rules [137]. This diversity helps to satisfy the personalized needs of the stakeholders who have different intents and expertise. We anticipate that this trend will persist, with the continued proposal and application of new XAI techniques to various SE tasks.

#### ► RQ<sub>2b</sub> - Summary ◀

- A variety of explanation formats have been explored in our surveyed XAI4SE research, with the main formats utilized being *numeric* (~22%), *text* (~19%), *visualization* (~14%), *source code* (~24%), and *rule* (~21%).

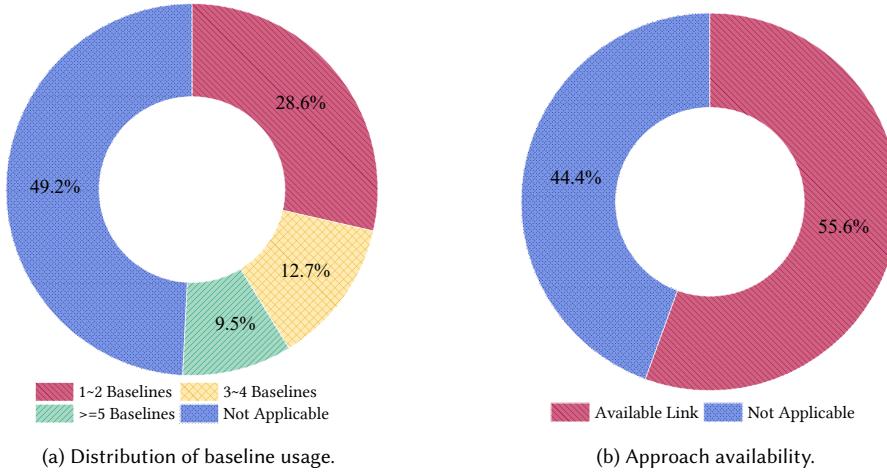


Fig. 13. Statistical information of baseline techniques and XAI4SE approaches.

- Source code is the most essential explanation format in XAI4SE studies. Few studies employed visualization as their preferred format of explanation due to the limited information it conveyed.
- The formats of explanations varied even in a single SE task. This diversity helps to satisfy the personalized needs of the stakeholders who have different intents and expertise.

## 6 RQ<sub>3</sub>: 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 the 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.

### 6.1 RQ<sub>3a</sub>: What baseline techniques are used to evaluate XAI4SE approaches?

For RQ<sub>3a</sub>, we scrutinize the baseline techniques employed for comparative analysis in our selected primary studies. Although common baselines for particular software engineering 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 paper compared against on our interactive website at <https://riss-vul.github.io/xai4se-paper/>. Fig. 13a briefly analyzes the distribution of baseline usage in XAI4SE studies. Approximately half of the papers reviewed do not engage in comparisons with any baseline ( $\approx 49.5\%$ ), whereas a minority contrasts their findings with more than four distinct methods ( $\approx 9.5\%$ ). 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

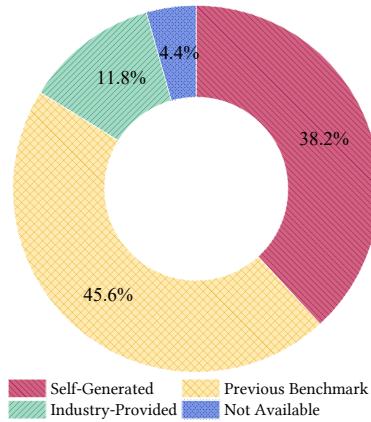


Fig. 14. Construction strategies of benchmarks.

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 high degree of specificity, varying significantly even among studies addressing identical software engineering tasks. For example, to evaluate the effectiveness of their proposed explainable vulnerability detection approaches, six out of 11 primary studies employed at least two baselines for evaluation, while only two papers [26, 129] 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 paper as a search query. As depicted in the pie chart in Fig. 13b, approximately only 44% 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 information search on Q&A sites [50] and OSS sustainability prediction [114]. 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 papers 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.

## 6.2 RQ<sub>3b</sub>: What benchmarks are used for these comparisons?

For RQ<sub>3b</sub>, we investigated the collection strategies of benchmarks in XAI4SE studies. As can be seen from the data in Fig. 14, 31 ( $\approx$ 45%) 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

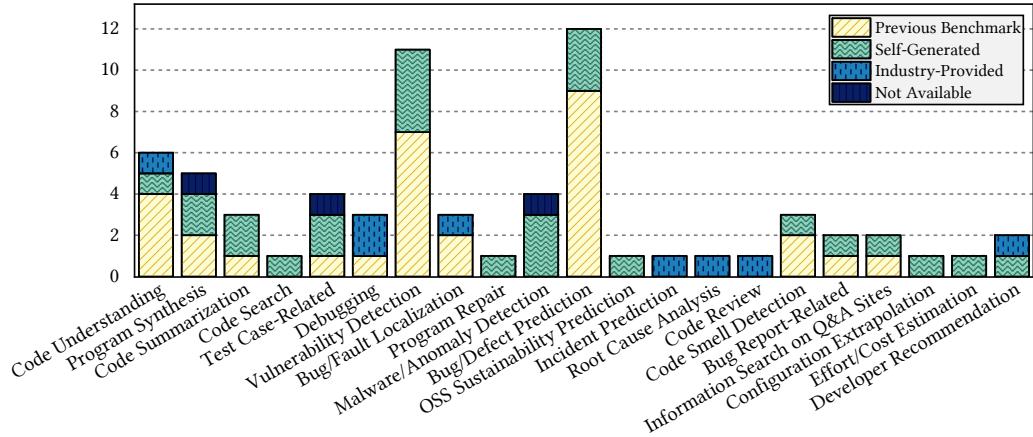


Fig. 15. Collection strategies of benchmarks by the SE task.

of XAI4SE research, there is an observed scarcity of appropriate benchmark datasets. This also explains why there is a considerable amount (38.2%) of self-generated 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 software engineering tasks. Furthermore, we 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 the benchmarks employed in these primary studies is depicted in Figure 15. For vulnerability detection, we found that the Big-Vul [22] dataset was used frequently, including evaluating the accuracy of the key aspects extracted from the detected vulnerabilities (e.g., vulnerable statements [36, 47], vulnerability types [26]). Additionally, in the context of defect prediction, the dataset released by Yatish et al. [120] was employed as a benchmark for comparing prior XAI techniques targeting software defect prediction [40, 44].

### 6.3 RQ<sub>3c</sub>: What evaluation metrics are employed to measure XAI4SE approaches?

With an increasing number of XAI techniques, the demand grows for suitable evaluation metrics [3, 5, 32]. 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 [79]. Table 16 describes the distribution of evaluation strategies found in this SLR. In our analysis of utilized evaluation strategies within work on XAI4SE, we observed that nearly half ( $\approx 44\%$ ) of primary studies adopted anecdotal evidence instead of quantitative metrics to evaluate their proposed approaches. That's to be expected because traditional performance metrics exist to evaluate prediction accuracy and computational complexity, while auxiliary criteria such as explainability may not be easily quantified [18]. Among 45 papers 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

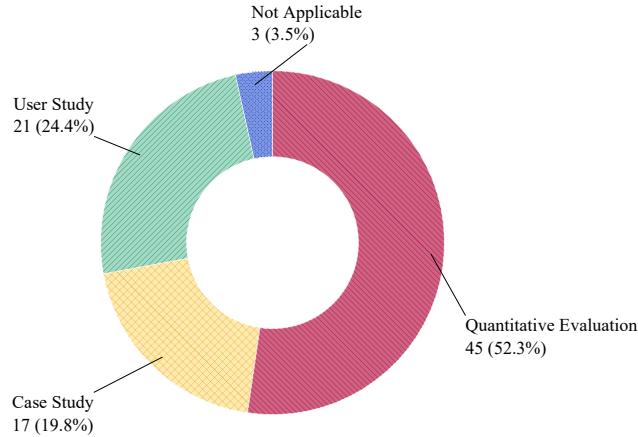


Fig. 16. Evaluation strategies of XAI techniques.

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. The evaluation metrics derived from an analysis of 63 papers have been organized into five distinct categories, i.e., metrics for *numerical explanations*, *textual explanations*, *visual explanations*, *code explanations*, and *rule explanations*, based on the various formats of explanations described above in Section 5.2. The detailed information is displayed in Table 9. For each explanation format, we listed the main evaluation metrics researched by no less than two studies.

**Numerical Explanation Metrics.** Commonly used metrics for numerical explanations are distance-based metrics and #Key Features (i.e., the number of key features). Both of them aim to quantify to what extent truly meaningful features are used by AI4SE models for decision. For instance, Sun et al. [91] employed Cosine Similarity to evaluate the transferability of adversarial cases with malicious features extracted by SHAP. Additionally, Liu et al. [51] leveraged Top-K features that positively contribute to the predictions of state-of-the-art malware detectors to validate whether they learned actual malicious behaviors for classification.

**Textual Explanation Metrics.** Metrics in this category originate from NLP tasks, measuring general quality aspects of the generated text. In the studies reviewed, the most widely used textual explanation metrics are Accuracy [26, 118], Precision [118, 135], Recall [118, 135], F1-score [26, 113, 118, 135], AUC [118, 135], BLEU [57, 129], and ROUGE [90, 129]. These metrics just correspond to two ways of achieving textual explanations: retrieving candidate answers from external knowledge bases (i.e., classification task) and generating from scratch by using generative models (i.e., generation task). For example, Fu et al. [26] employed Accuracy and Weighted F1-score to evaluate the performance of vulnerability classification. Similarly, Yang et al. [118] adopted Accuracy, Precision, Recall, and F1-score to assess the performance of each text classifier from the attribute perspective. To measure the accuracy of explainable silent dependency alert generation, Sun et al. [90] used ROUGE-1, ROUGE-2, and ROUGE-L as evaluation metrics.

**Visual Explanation Metrics.** For visual explanations, attention weight is the most frequent metric, used in code understanding [99], vulnerability detection [52], and code smells [102]. In addition, statistic-based metrics, such as Pearson Correlation [39] and Spearman Correlation [99], are also utilized to measure the quality of generated explanations. Intuitively, if an explanation approach is more reasonable, its generated explanations should be more semantically related to the sample to be explained.

Table 9. Metrics Used for Evaluation

Format	Metric	#Studies	Reference
Numeric	Distance-Based	2	[1, 91]
	#Key Features	3	[14, 44, 51]
Text	Accuracy	2	[26, 118]
	Precision	2	[118, 135]
	Recall	2	[118, 135]
	F1-Score	4	[26, 113, 118, 135]
	AUC	2	[118, 135]
	BLEU	2	[57, 129]
Visualization	Attention Weights	3	[52, 99, 102]
	Statistic-Based	2	[39, 99]
Source Code	Accuracy	3	[36, 47, 117]
	Recall	2	[93, 107]
	F1-Score	2	[71, 117]
	BLEU	2	[30, 79]
	Top K-Based	6	[75, 103, 107, 110, 117, 130]
	Statistic-Based	2	[36, 79]
	Mean First Rank (MFR)	2	[47, 130]
	Mean Average Rank (MAR)	2	[47, 130]
Rule	Accuracy	3	[8, 48, 137]
	Precision	3	[8, 31, 58]
	Recall	2	[8, 31]
	F1-Score	5	[29, 31, 74, 84, 137]
	AUC	2	[29, 74]
	Statistic-Based	2	[74, 76]

**Code Explanation Metrics.** For code explanations, traditional classification metrics (e.g., Accuracy [36, 47, 117], Recall [93, 107], and F1-score [29, 31, 74, 84, 137]), and ranking-based metrics like Top- $k$  Accuracy [75, 103, 107, 110, 117, 130] are the most commonly used. For example, Yang et al. [117] combined multiple metrics, including Top-1 Recall, Top-5 Recall, Accuracy, and F1-score to measure the contributions of interpretable tensor propagation to improving the performance of variable misuse detection.

**Rule Explanation Metrics.** In terms of rule explanations, the metrics need to measure to what extent the generated rules can cover and correctly explain the target data. There are five frequently used performance metrics, including Accuracy [8, 48, 137], Precision [8, 31, 58], Recall [8, 31], F1-score [29, 31, 74, 84, 137], and AUC [29, 74], for evaluating the quality of rule explanations. For example, Zou et al [137] employed Accuracy and F1-score to evaluate the fidelity of different explanation approaches. Cito et al. [8] and Gesi et al. [31] used Precision, Recall, and F1-score to measure the coverage of generated rules for mispredictions.

#### ▶ RQ<sub>3</sub> - Summary ◀

- The analysis indicated a notable scarcity of well-documented and reusable baselines or benchmarks for work on XAI4SE. Approximately 49% of the baseline techniques employed in the evaluations of our studied XAI4SE 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 innovative and task-specific metrics or researchers' subjective intuition of what constitutes a good explanation.
- The evaluation metrics predominantly utilized across the studies were categorized based on five types of explanation formats, i.e., numeric, text, visualization, source code, and rule. A significant portion of the reviewed papers employed conventional classification metrics, such as Accuracy and F1-score, to evaluate the performance of their proposed approaches.

## 7 LIMITATIONS

**Study Collection Omission.** Our review has some potential limitations, and one of them is the risk of inadvertently excluding relevant studies during the literature search and selection phase. The incomplete summarization of keywords related to SE tasks and the varied use of terminology for explainability across studies may have led to our search criteria overlooking relevant research that ought to have been incorporated into our SLR. To address this concern, we first manually selected 26 top-tier SE & AI venues suggested by previous surveys on AI4SE research [101, 106, 119], and extracted relatively comprehensive and standard keywords for SE tasks and XAI techniques. With these search strings, we further augmented our search results by combining automated search with forward-backward snowballing.

**Data Extraction Bias.** Another potential limitation is data extraction bias. Certain discrepancies arose inevitably when extracting related content and classifying the data items in Table 6. To mitigate the bias in data extraction phase to the validity of our findings, we invited two practitioners, the fourth and 11th authors, to conduct a secondary review of controversial data items that unable to reach consensus on classification. Both of them have more than 10 years of experience in the field of SE and XAI.

By applying these countermeasures, we strive to guarantee the comprehensiveness of the selected papers and the accuracy of the data items, thereby enhancing the reliability of our findings.

## 8 CHALLENGES AND OPPORTUNITIES

In this section, we discuss current challenges (Section 8.1) and highlight promising opportunities (Section 8.2) for applying XAI techniques in AI4SE research.

### 8.1 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 [107], while others extract human-understandable rules [74], or even natural language descriptions [57] 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: Trade-off between Performance and Explainability.** For some real-world tasks, a model with higher accuracy usually offers less explainability (and vice versa). Such *Performance-Explainability Trade-off (PET)* dilemma often results in user hesitation when choosing between black-box models and inherently transparent models. While certain studies [80, 81] 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 the characteristics (e.g., input/output format, model architecture) of different SE tasks. For example, giving precedence to transparent models when their performance could meet the users' requirements, and transitioning to more complex models when needed.

**Challenge 3: Highly Individualized Baselines and Benchmarks.** Evaluating the newly proposed approach over baseline techniques on a benchmark dataset is developing into a standard practice in SE research. Nevertheless, ground truth information scarcity is still a major issue since 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, 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 [45] for evaluating the explainability on medical report generation. Additionally, only a small subset of primary studies attempt to perform a comparison over baseline techniques. This is possibly due to the lack of publicly available and replicable baselines, as well as the difference between each approach's explanation format. For example, it is evident from Table 12, the explanation formats vary even in a single vulnerability detection task, including text (e.g., vulnerability descriptions [68, 129], types [90, 135]), visualization (e.g., heatmap [52]), source code (e.g., code statements [47, 93]), and rules [137]. The lack of standardized baselines can lead to confusing or conflicting results, i.e., the performance score of the same baseline technique varies in different papers. To exclude manual efforts and subjective bias, we highlight the importance of replicability and reproducibility for XAI4SE studies. The researchers should share replication packages and provide sufficient details when describing the processes of data preparation, approach implementation, and performance evaluation.

**Challenge 4: Limited Performance Measures.** The last open challenge is the limited evaluation of the explanations provided by XAI techniques. The most adopted approach is to resort to the practitioners' expertise, i.e., relying on the researchers' intuition of what constitutes a good explanation [60]. As previously discussed in Section 6.3, nearly half ( $\approx 44\%$ ) of XAI4SE studies adopted anecdotal evidence instead of quantitative metrics to evaluate their proposed approaches. However, considering the variability in experts' opinions, this strategy is particularly biased and subjective [66]. 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.

In addition, various metrics have been employed for quantitative evaluation, some special concerns within the XAI field are not fully considered yet:

- **Efficiency.** 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.

- **Other Important Metrics.** Explainability is a multi-faceted concept. Thus, instead of evaluating only one property of explanation quality (e.g., *Explanation Accuracy* which measures how consistent the explanation results are with the researchers' intuition [47, 117]), it is essential to get insight into, and preferably quantify, all properties such that an informed trade-off can be made [66]. For example, apart from effectiveness, Hu et al. [36] also evaluated the robustness, stability, and consistency of state-of-the-art explanation approaches for GNN-based vulnerability detectors. 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, as the Large Language Model (LLM) community recently does [72].

## 8.2 Opportunities

**Opportunity 1: Applying XAI to Other SE tasks.** We observed that XAI techniques have been widely used in certain SE tasks to support users in decision-making or improve the transparency of AI models, such as defect prediction, vulnerability detection, code understanding, and so on. However, the current application of XAI techniques in some SE activities remains relatively sparse. As shown in Fig. 6, not many studies focus on software development and management. This unveils a substantial opportunity: broadening the application of XAI techniques to these under-explored research topic. Such an extension could lead to more practical and actionable AI-driven solutions across the entire SDLC.

**Opportunity 2: Developing XAI Suitable for SE Tasks.** We noted 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 domain knowledge appears to be the most promising direction to explore.

**Opportunity 3: Human-in-the-Loop Interaction.** 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 [108]. Among all papers reviewed, Wang et al. [100] leveraged a structured conceptual tree to display the results of query scoping, the identified conceptual association paths. Users can interact by selecting the anchor concept relevant to the query which will update the results of code snippets related to the selected concepts.

**Opportunity 4: Deployment in Practice.** The deployment of XAI techniques in SE practice, particularly security-critical tasks like vulnerability detection, necessitates rigorous validation to satisfy not only effectiveness, but also robustness, controllability, and other special concerns. 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 example, visual explanation can be attractive for the layperson since it is natural and intuitive. Nonetheless, complex visualization tools could potentially overwhelm domain experts with superfluous information, thereby increasing their cognitive load and rendering the tool counterproductive [121]. Therefore, a

potential opportunity is to work closely with relevant SE practitioners to continuously iterate the XAI tool based on their feedback.

## 9 CONCLUSION

Explainability remains a pivotal area of interest within the SE community, particularly as increasingly advanced AI models rapidly advance the field. This paper conducts a systematic literature review of 63 primary studies on XAI4SE research from top-tier SE and AI conferences and journals. Initially, we formulated a series of research questions 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 research roadmap that delineates 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] Jubril Gbolaran 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.
- [2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald C. Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: a case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE/ACM, 291–300.
- [3] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion* 58 (2020), 82–115.
- [4] 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.
- [5] Nadia Burkart and Marco F. Huber. 2021. A Survey on the Explainability of Supervised Machine Learning. *J. Artif. Intell. Res.* 70 (2021), 245–317.
- [6] Shyam R. Chidamber and Chris F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.* 20, 6 (1994), 476–493.
- [7] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülgahre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 19th Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 1724–1734.
- [8] 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.
- [9] 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.
- [10] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.
- [11] 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.
- [12] Raphaël Dang-Nhu. 2020. PLANS: Neuro-Symbolic Program Learning from Videos. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*.
- [13] 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.
- [14] 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.

- [15] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, Hao Fan, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 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.
- [16] 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.
- [17] Geanderson Esteves dos Santos, Eduardo Figueiredo, Adriano Veloso, Markos Viggiani, and Nivio Ziviani. 2020. Understanding machine learning software defect predictions. *Autom. Softw. Eng.* 27, 3 (2020), 369–392.
- [18] 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.
- [19] Finale Doshi-Velez and Been Kim. 2018. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv: 1702.08608* (2018).
- [20] Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Omer F. Rana, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, and Rajiv Ranjan. 2023. Explainable AI (XAI): Core Ideas, Techniques, and Solutions. *ACM Comput. Surv.* 55, 9 (2023), 194:1–194:33.
- [21] 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.
- [22] 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.
- [23] Julian J Faraway. 2004. *Linear models with R*. Chapman and Hall/CRC.
- [24] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Dixin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv preprint arXiv: 2002.08155* (2020).
- [25] Jerome H Friedman and Bogdan E Popescu. 2008. Predictive learning via rule ensembles. (2008).
- [26] Michael Fu, Van Nguyen, Chakkrit Tantithamthavorn, Trung Le, and Dinh Phung. 2023. VulExplainer: A Transformer-Based Hierarchical Distillation for Explaining Vulnerability Types. *IEEE Trans. Software Eng.* 49, 10 (2023), 4550–4565.
- [27] Michael Fu and Chakkrit Tantithamthavorn. 2023. GPT2SP: A Transformer-Based Agile Story Point Estimation Approach. *IEEE Trans. Software Eng.* 49, 2 (2023), 611–625.
- [28] 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.
- [29] Yuxiang Gao, Yi Zhu, and Yu Zhao. 2022. Dealing with imbalanced data for interpretable defect prediction. *Inf. Softw. Technol.* 151 (2022), 107016.
- [30] 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.
- [31] 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.
- [32] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv.* 51, 5 (2019), 93:1–93:42.
- [33] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 7212–7225.
- [34] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Dixin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*.
- [35] Md Imran Hossain, Ghada Zamzmi, Peter R. Mouton, Md Sirajus Salekin, Yu Sun, and Dmitry Goldgof. 2023. Explainable AI for Medical Data: Current Methods, Limitations, and Future Directions. *ACM Comput. Surv.* (2023).
- [36] 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.
- [37] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).
- [38] 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.
- [39] 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 Trans. Software Eng.* 49, 4 (2023), 2239–2254.
- [40] Jirayut Jiarpakdee, Chakkrit Kla Tantithamthavorn, Hoa Khanh Dam, and John C. Grundy. 2022. An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models. *IEEE Trans. Software Eng.* 48, 2 (2022), 166–185.
- [41] 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.

- [42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NeurIPS)*. 1106–1114.
- [43] 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.
- [44] 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.
- [45] Mingjie Li, Wenjia Cai, Rui Liu, Yuetian Weng, Xiaoyun Zhao, Cong Wang, Xin Chen, Zhong Liu, Caineng Pan, Mengke Li, Yingfeng Zheng, Yizhi Liu, Flora D. Salim, Karin Verspoor, Xiaodan Liang, and Xiaojun Chang. 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)*.
- [46] 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.
- [47] 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.
- [48] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wench Zhang, Kaixin Sui, Yanhua Wang, Xu Du, Guoqiang Duan, and Dan Pei. 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.
- [49] 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 Trans. Software Eng.* 48, 4 (2022), 1268–1289.
- [50] 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.
- [51] 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.
- [52] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinning 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.
- [53] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 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*.
- [54] 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. *Nat. Mach. Intell.* 2, 1 (2020), 56–67.
- [55] 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.
- [56] Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. CoCoNuT: combining context-aware neural translation models using ensemble for program repair. In *Proceedins of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 101–114.
- [57] 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.
- [58] 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.
- [59] Pablo Messina, Pablo Pino, Denis Parra, Alvaro Soto, Cecilia Besa, Sergio Uribe, Marcelo E. Andia, Cristian Tejos, Claudia Prieto, and Daniel Capurro. 2022. A Survey on Deep Learning and Explainability for Automatic Report Generation from Medical Images. *ACM Comput. Surv.* 54, 10s (2022), 203:1–203:40.
- [60] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.* 267 (2019), 1–38.
- [61] 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. *arXiv preprint arXiv: 2302.06065* (2023).
- [62] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognit.* 65 (2017), 211–222.
- [63] Toshiki Mori and Naoshi Uchihira. 2019. Balancing the trade-off between accuracy and interpretability in software defect prediction. *Empir. Softw. Eng.* 24, 2 (2019), 779–825.

- [64] Tamara Munzner. 2014. *Visualization analysis and design*. CRC press.
- [65] 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.
- [66] 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 Comput. Surv.* 55, 13s (2023), 295:1–295:42.
- [67] 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.
- [68] 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.
- [69] OpenAI. 2022. Chatgpt: Optimizing language models for dialogue. <https://chat.openai.com>.
- [70] Ipek Ozkaya and James Ivers. 2019. AI for Software Engineering. In *Proceedings of the SEI Educator's Workshop*.
- [71] 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.
- [72] Daniel Park. 2023. *Open-LLM-Leaderboard-Report*. <https://github.com/dsdanielpark/Open-LLM-Leaderboard-Report>
- [73] Cristiano Patrício, João C. Neve, and Luis F. Teixeira. 2023. Explainable Deep Learning Methods in Medical Image Classification: A Survey. *ACM Comput. Surv.* 56, 4 (2023), 85:1–85:41.
- [74] 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.
- [75] 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.
- [76] Dilini Rajapaksha, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Christoph Bergmeir, John Grundy, and Wray L. Buntine. 2022. SQAPlanner: Generating Data-Informed Software Quality Improvement Plans. *IEEE Trans. Software Eng.* 48, 8 (2022), 2814–2835.
- [77] 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 Trans. Softw. Eng. Methodol.* 28, 3 (2019), 15.
- [78] 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.
- [79] 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.
- [80] 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.
- [81] 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.
- [82] Nayan B. Ruparelia. 2010. Software development lifecycle models. *ACM SIGSOFT Softw. Eng. Notes* 35, 3 (2010), 8–13.
- [83] 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. *Int. J. Comput. Vis.* 128, 2 (2020), 336–359.
- [84] 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. *Empir. Softw. Eng.* 18, 5 (2013), 1005–1042.
- [85] 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)*, Vol. 70. PMLR, 3145–3153.
- [86] 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.
- [87] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. 2020. explAIner: A Visual Analytics Framework for Interactive and Explainable Machine Learning. *IEEE Trans. Vis. Comput. Graph.* 26, 1 (2020), 1064–1074.
- [88] M. Srinivas and Lalit M. Patnaik. 1994. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Trans. Syst. Man Cybern.* 24, 4 (1994), 656–667.
- [89] Mateusz Staniak and Przemyslaw Biecek. 2018. Explanations of Model Predictions with live and breakDown Packages. *R J.* 10, 2 (2018), 395.
- [90] 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.

- [91] 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.
- [92] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, Vol. 70. PMLR, 3319–3328.
- [93] 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 Trans. Softw. Eng. Methodol.* 32, 6 (2023), 145:1–145:40.
- [94] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NeurIPS)*. 3104–3112.
- [95] 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 Press / The MIT Press, 900–907.
- [96] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*. 5998–6008.
- [97] 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)*.
- [98] Yao Wan, Yang He, Zhangqian Bi, Jianguo Zhang, Hongyu Zhang, Yulei Sui, Guandong Xu, Hai Jin, and Philip S. Yu. 2023. Deep Learning for Code Intelligence: Survey, Benchmark and Toolkit. *arXiv preprint arXiv: 2401.00288* (2023).
- [99] 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.
- [100] 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 Trans. Softw. Eng. Methodol.* 32, 6 (2023), 140:1–140:28.
- [101] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. 2023. Machine/Deep Learning for Software Engineering: A Systematic Literature Review. *IEEE Trans. Software Eng.* 49, 3 (2023), 1188–1231.
- [102] 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.
- [103] 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.
- [104] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 26th Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 8696–8708.
- [105] Yuekun Wang, Yuhang Ye, Yueming Wu, Weiwei Zhang, Yinxing Xue, and Yang Liu. 2023. Comparison and Evaluation of Clone Detection Techniques with Different Code Representations. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 332–344.
- [106] Cody Watson, Nathan Cooper, David Nader-Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.* 31, 2 (2022), 32:1–32:58.
- [107] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Chakkrit Tanithamthavorn, Hideaki Hata, and Kenichi Matsumoto. 2022. Predicting Defective Lines Using a Model-Agnostic Technique. *IEEE Trans. Software Eng.* 48, 5 (2022), 1480–1496.
- [108] 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.
- [109] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 87–98.
- [110] Ratnadira Widayarsi, Gede Artha Azriadi Prana, Stefanus A. Haryono, Yuan Tian, Hafli Noer Zachairy, 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.
- [111] Jan De Winter. 2010. Explanations in Software Engineering: The Pragmatic Point of View. *Minds Mach.* 20, 2 (2010), 277–289.
- [112] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. ACM, 38:1–38:10.
- [113] 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 Trans. Softw. Eng. Methodol.* 30, 2 (2021), 21:1–21:29.
- [114] 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.
- [115] Xinqiang Xie, Xiaochun Yang, Bin Wang, and Qiang He. 2022. DevRec: Multi-Relationship Embedded Software Developer Recommendation. *IEEE Trans. Software Eng.* 48, 11 (2022), 4357–4379.

- [116] 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.
- [117] Jia Yang, Cai Fu, Fenyang Deng, Ming Wen, Xiaowei Guo, and Chuanhao Wan. 2023. Toward Interpretable Graph Tensor Convolution Neural Network for Code Semantics Embedding. *ACM Trans. Softw. Eng. Methodol.* 32, 5 (2023), 115:1–115:40.
- [118] 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.
- [119] Yanming Yang, Xin Xia, David Lo, and John C. Grundy. 2022. A Survey on Deep Learning for Software Engineering. *ACM Comput. Surv.* 54, 10s (2022), 206:1–206:73.
- [120] 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)*. IEEE / ACM, 654–665.
- [121] Wei Jie Yeo, Wihan van der Heever, Rui Mao, Erik Cambria, Ranjan Satapathy, and Gianmarco Mengaldo. 2023. A Comprehensive Review on Financial Explainable AI. *arXiv preprint arXiv: 2309.11960* (2023).
- [122] 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.
- [123] 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.
- [124] Hao Yu, Yiling Lou, Ke Sun, Dezhui 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.
- [125] Éloi Zablocki, Hédi Ben-Younes, Patrick Pérez, and Matthieu Cord. 2022. Explainability of Deep Vision-Based Autonomous Driving Systems: Review and Challenges. *Int. J. Comput. Vis.* 130, 10 (2022), 2425–2452.
- [126] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Proceedings of the 13th European Conference on Computer Vision (ECCV) (Lecture Notes in Computer Science, Vol. 8689)*. Springer, 818–833.
- [127] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. *IEEE Trans. Software Eng.* 28, 2 (2002), 183–200.
- [128] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying relevant studies in software engineering. *Inf. Softw. Technol.* 53, 6 (2011), 625–637.
- [129] 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.
- [130] 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.
- [131] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 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.
- [132] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, Kaixin Sui, and Dan Pei. 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.
- [133] Wei Zheng, Tianren Shen, Xiang Chen, and Peiran Deng. 2022. Interpretability application of the Just-in-Time software defect prediction model. *J. Syst. Softw.* 188 (2022), 111245.
- [134] Bolei Zhou, Aditya Khosla, Ágata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning Deep Features for Discriminative Localization. In *Proceedings of the 26 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2921–2929.
- [135] 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.
- [136] Julia El Zini and Mariette Awad. 2023. On the Explainability of Natural Language Processing Deep Models. *ACM Comput. Surv.* 55, 5 (2023), 103:1–103:31.
- [137] 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 Trans. Softw. Eng. Methodol.* 30, 2 (2021), 23:1–23:31.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009