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

SICONG CAO, School of Information Engineering, Yangzhou University, Yangzhou, China and Industry-Education Integration Innovation Center of Specialized Cybersecurity (Yangzhou University), Yangzhou, China
XIAOBING SUN*, School of Information Engineering, Yangzhou University, Yangzhou, China
RATNADIRA WIDYASARI, Singapore Management University, Singapore, Singapore
DAVID LO, Singapore Management University, Singapore, Singapore
XIAOXUE WU, School of Information Engineering, Yangzhou University, Yangzhou, China
LILI BO, School of Information Engineering, Yangzhou University, Yangzhou, China and Yunnan Key Laboratory of Software Engineering, Kunming, China
JIALE ZHANG, School of Information Engineering, Yangzhou University, Yangzhou, China
BIN LI, School of Information Engineering, Yangzhou University, Yangzhou, China
WEI LIU, School of Information Engineering, Yangzhou University, Yangzhou, China
DI WU, School of Mathematics, Physics, and Computing, University of Southern Queensland, Toowoomba, Australia
YIXIN CHEN, Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, USA

The online appendix introduces the review methodology of our main paper, which is outlined as follows. Appendix A.1 describes the search strategy to identify relevant studies. Appendix A.2 presents the procedure to select the primary studies that provide direct evidence about the research questions. Appendix A.3 provides the basic review results, including the publication statistics over years and distribution of publications in various venues. Appendix A.4 discusses the possible threats to validity in our review process. Appendix A.5 shows the usages of XAI techniques in specific SE tasks.

*Corresponding author

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

0360-0300/2025/8-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

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

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

ACM Reference Format:
Sicong Cao, Xiaobing Sun, Ratnadira Widyasari, David Lo, Xiaoxue Wu, Lili Bo, Jiale Zhang, Bin Li, Wei Liu, Di Wu, and Yixin Chen. 2025. A Systematic Literature Review on Explainability for Machine/Deep Learning-based Software Engineering. *ACM Comput. Surv.* 1, 1 (August 2025), 15 pages. https://doi.org/XXXXXXX.XXXXXXX

A REVIEW METHODOLOGY

A.1 Search Strategy

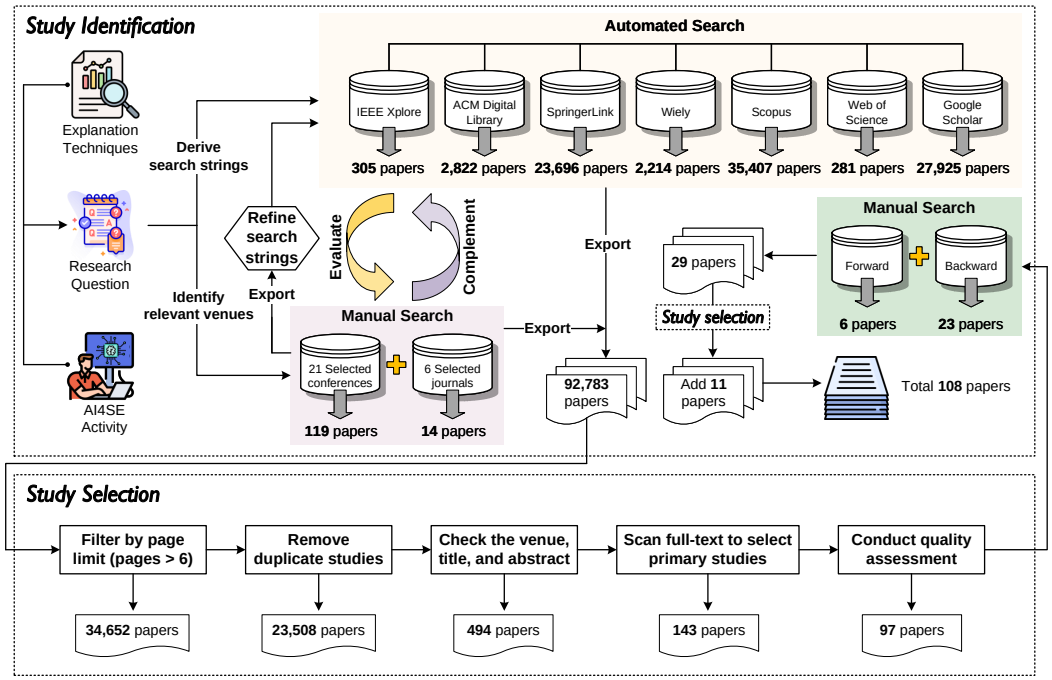


Fig. 1. Study identification and selection process.

As shown in Figure 1, following the standardized practice within the field of SE [45], 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 in 2012 – has transformed AI research and became the catalyst for the ML/DL boom in all fields including SE, we chose a 13-year period of January 1st, 2012, to December 31st, 2024, 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 1. In total, we included 16 conferences (ICSE, ASE, ESEC/FSE, ICSME, ICPC, RE, ESEM, ISSTA, MSR, SANER, ISSRE, APSEC, 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)

Table 1. Publication Venues for Manual Search

Venue	Acronym	Full name
Conference	ICSE	IEEE/ACM 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
	APSEC	Asia-Pacific Software Engineering Conference
	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

* The conference name is changed to ACM International Conference on the Foundations of Software Engineering (FSE) since 2024.

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[1], ACM Digital Library[2], SpringerLink[3], Wiely[4], and Scopus[5], and two of the most popular research citation engines, Web of Science[6] and Google Scholar[7], based on the search string (listed in Table 2) assembled from a group of topic-related keywords summarized from manually collected papers. As shown in Table 3, we collected a total of 92,783 relevant studies with the automatic search from these seven electronic databases.

[1]https://ieeexplore.ieee.org
[2]https://dl.acm.org
[3]https://link.springer.com
[4]https://onlinelibrary.wiley.com
[5]https://www.scopus.com
[6]https://www.webofscience.com
[7]https://scholar.google.com

Table 2. Search Keywords

Group	Keywords
1	*"Machine Learn*"* 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*"* OR *"Software Evolution"* OR *"Software Test*"* OR *"Software Requirement?"* OR *"Software Develop*"* OR *"Project Management"* OR *"Software Design*"* 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.

Table 3. Summary of the Process of Study Search and Selection

Data Source	# Studies
IEEE Xplore	305
ACM Digital Library	2,822
SpringerLink	23,696
Wiely	2,214
Scopus	35,407
Web of Science	281
Google Scholar	27,925
Merge	92,783
Filtering studies less than 6 pages	34,652
Removing duplicated studies	23,508
Excluding primary studies based on venue, title, and abstract	494
Excluding primary studies based on full text	143
After Quality Assessment	97
After Forward & Backward Snowballing	126
Final	108

A.2 Study Selection

A.2.1 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.*

Table 4. Checklist of Quality Assessment Criteria for Explainability Studies in AI4SE

No.	Quality Assessment Criteria
QAC_1	Is the impact of the proposed approach (or empirical/case study) on the AI4SE community clearly stated?
QAC_2	Are the contributions of the study clearly claimed?
QAC_3	Does the study provide a clear description of the workflow and implementation of the proposed approach?
QAC_4	Are the experiment details, including datasets, baselines, and evaluation metrics, clearly described?
QAC_5	Do the findings drawn from the experiments strongly substantiate the arguments presented in the study?

✓ *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 are excluded.*

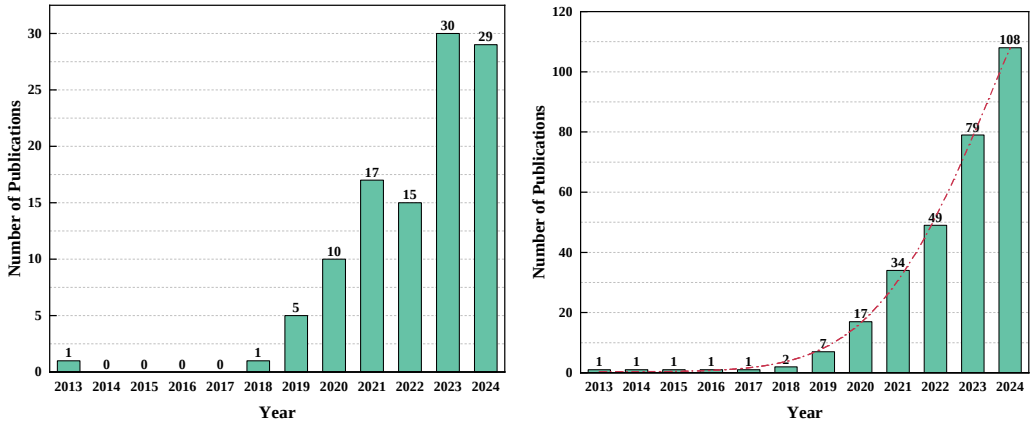
In particular, by literature filtering and deduplication (exclusion criteria 1), the total number of included papers was reduced to 23,508. After the first two authors manually examined the venue, title, and abstracts of the papers, the total number of included papers declined substantially to 494. Any 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 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 143 primary studies directly relevant to our research topic.

A.2.2 Quality Assessment. To prevent biases introduced by low-quality studies, we formulated five **Quality Assessment Criteria (QAC)**, given in Table 4, to evaluate the 143 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 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 113 primary studies. After quality assessment, a final set of 97 high-quality papers was reserved.

A.2.3 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, 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 97 high-quality papers. As a supplement, we gathered 29 more papers, and

Table 5. Extracted Data Items and Related Research Questions

RQ	Data Item
RQ_1	The SE task that an XAI4SE approach tries to solve
RQ_1	The SE activity in which each SE task belongs
RQ_1	Publication type of each primary study (i.e., new technique, empirical study, or case study)
RQ_2	XAI technique employed by each study
RQ_2	Explanation format
RQ_3	The adopted baseline approaches
RQ_3	Benchmark dataset name
RQ_3	Presence/absence of replication package
RQ_3	What metrics are used to evaluate the XAI techniques



(a) Distribution of publications per year.

(b) Cumulative publication trend over years.

Fig. 2. Publication statistics over years.

conducted the complete study selection process again, including filtering, deduplication, and quality assessment, and obtained 11 additional papers.

A.3 Data Extraction and Analysis

Based on the collected 108 primary studies, we extracted the essential data items used to answer three main RQs. In Table 5, we outline the details information extracted and gathered from 108 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.

Figure 2a presents the distribution of selected primary studies in each year. The first XAI4SE study we found was published in 2013. After that, there was a 4-year research gap, ranging from 2014 to 2017. The enthusiasm for investigating the explainability on AI4SE models has steadily risen since 2018, and reaches its peak in recent two years, comprising 54.6% of the total publications. Figure 2b 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 2024. This pronounced upward trend indicates a burgeoning research interest in the field of XAI4SE.

(a) Distribution of publications in various conferences.

(b) Distribution of publications in various journals.

Fig. 3. Distribution of selected papers in different publication venues.

We also analyzed the publication trend of primary studies in selected conferences and journal venues, respectively. As shown in Figure 3a, ESEC/FSE stands out as the predominant conference venues favored by XAI4SE studies, with a contribution of 26.8% of the total. Other venues making noteworthy contributions include ICSE (14.1%), ASE (12.7%), and SANER/QRS (5.6%). Figure 3b shows the distribution of primary papers published in different journal venues. It can be seen that 75.6% 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.

A.4 Threats to Validity

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 27 top-tier SE & AI venues suggested by previous surveys on AI4SE research [33, 34, 41], 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 5. 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.

A.5 How XAI Are Used in Specific SE Tasks?

A.5.1 SE Tasks in Software Requirements & Design. Software requirements refer to specific descriptions of conditions or capabilities needed by users, systems, or system components, while software design involves the process of defining the structure, components, functionalities, interfaces, and

their relationships within a software system. During this phase, only one topic, i.e., *Requirement Classification*, is explored, leaving ample space for further exploration.

Requirement Classification. As a key example of ML applied to requirements engineering, requirement classification aims to categorize software requirements into different classes or types, such as functional and non-functional requirements. Dalpiaz et al. [7] constructed ML classifiers based on more general linguistic features (e.g., dependency types), and leveraged modern rule-based XAI tools to identify those features that appeared commonly and that helped distinguish functional and quality aspects.

A.5.2 SE Tasks in Software Development. There are wide-ranging applications of XAI techniques in software development, encompassing tasks such as *Code Understanding*, *Program Synthesis*, and *Code Summarization*.

Code Understanding. Code understanding refers to the process of comprehending and analyzing source code deeply. Within the context of data-driven SE research, code understanding aims to seek an effective way to map source code into high-dimensional semantic space, thereby supporting a variety of code-centric downstream tasks. Inspired by the capability of complex AI models, deep neural networks in particular, in learning rich representations of raw data, a series of code models are trained on labeled (e.g., CodeSearchNet [16]) or unlabeled code corpus (e.g., CodeXGlue [22]). This training process produces code embeddings with rich contexts and semantics. Yang et al. [40] 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. [31] 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.

Program Synthesis. Program synthesis refers to the automated process of generating source code or software programs to meet specified requirements based on specified specifications or requirements. Although DL-based program synthesis applications, such as CodeGeeX[8] and GitHub Copilot[9], have integrated into the daily workflow of software developers around the globe due to their high, even human-competitive accuracy, lacking transparency makes the automatically generated programs untrustworthy. To explain the fundamental working mechanism, Chen et al. [4] proposed a causality-inspired approach, which constructed a dependency graph to capture not only the natural dependency between input and output tokens, but the dependency among output tokens as well. Then, they leveraged causal inference to quantify the contribution of each dependency edge to the end prediction result. Besides, Nazari et al. [25] introduced subspecifications as a general mechanism to identify the constraints imposed by the global specification on individual parts of the implementation as explanatory notes.

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 Machine Translation (NMT)** [5, 30]. Despite their effectiveness, such end-to-end summarization without any explanation is still far from being usable in practice. To this end, Geng et al. [14] 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 the

[8]https://codegeex.cn/en-US
[9]https://copilot.microsoft.com/

each code token. Jiang et al. [17] 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.

Others. Apart from the above tasks, a number of studies applied XAI techniques on other research topics for improving AI-assisted software development. For instance, code search aims to retrieve source code that meets users' natural language queries from a large codebase. Wang et al. [32] 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. Huang et al. [15] designed a novel knowledge-aware HumanAI dialogue agent to interact with developers and return APIs with relevance explanation and extended knowledge.

A.5.3 SE Tasks in Software Testing. Within the context of software testing, we found versatile applications of XAI techniques across a spectrum of tasks, including *Test Case-Related Automation*, *Debugging*, *Vulnerability Detection*, and *Bug/Fault localization*.

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 [44], test case recommendation [18], and test case diversity analysis [1], has received widespread attention. Yu et al. [44] 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. [18] 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. [6] 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. [19] leveraged GNNExplainer [43] 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 [12], root cause [29], similar vulnerability reports [26], and so on. Zhou et al. [47] 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. [35] formulated the localization task as a binary classification problem, i.e., predicting whether a test case will fail or pass. They applied TreeSHAP [23], 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. [20] 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.

A.5.4 SE Tasks in Software Maintenance. Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs. The applications of XAI in software maintenance are diverse, including *Malware/Anomaly Detection*, *Bug/Defect Prediction*, *Root Cause Analysis*, *Code Smell Detection*, *Bug Report-Related Automation*, etc.

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. [36] 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[10], and leveraged the mapping relation between the malicious behaviors and their corresponding semantics to generate descriptions for malware.

Bug/Defect Prediction. In the past few years, defect prediction is the most extensive and active research topic in software maintenance. According to different granularities, these studies can be further classified into two categories: file-level and commit-level (also known as **Just-In-Time (JIT)**) defect prediction. File-level defect prediction techniques often employ a set of hand-crafted feature metrics extracted from a software product to construct the classification model. For instance, Yang et al. [39] proposed a weighted association rule based on the contribution degree of features to optimize the process of rule generation, ranking, pruning, and prediction. By contrast, JIT defect prediction task aims to help developers prioritize their limited energy and resources on the most risky commits that are most likely to introduce defects. Zheng et al. [46] trained a random forest

[10]https://developer.android.google.cn/docs

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.

Root Cause Analysis. Despite the broad adoption of microservice architecture in modern production systems, it is inherently susceptible to critical production incidents, underscoring the importance of swiftly identifying and addressing the root causes of any issues to ensure quick system recovery. Prior studies directly fed multi-modal data to a deep neural network, suffering from limited interpretability and operability. Yao et al. [42] analyzed causal relationships of events transformed from multi-modal observation data, and provided interpretable parameters with clear physical meanings that align well with the operational experience of site reliability engineers, thereby facilitating the integration of their expertise directly into the analysis process through human feedback. Ding et al. [9] leveraged reinforcement learning to learn an interpretable pruning policy for the service dependency graph to automatically eliminates redundant components. The policy is based on graph pruning rules derived from experienced engineers and comprehensive trace analysis, ensuring domain knowledge and industry best practices are incorporated.

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. [27] 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 [3]. For example, instead of using other black-box models, Shihab et al. [28] 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. [8] applied SHAP to compute the importance of tell smell features.

Others. Some primary studies explored the explainability of certain special SE tasks. For example, Markovtsev et al. [24] proposed a decision tree-based explainable model to express the found format patterns with compact human-readable rules. Xiao et al. [37] leveraged the local explanations generated by LIME from the XGBoost model to analyze the contribution of variables to sustained activity in different project contexts.

A.5.5 SE Tasks in Software Management. There are four literature involving the utilization of XAI in software management, involving the following main SE tasks, i.e., *Mining Software Repositories.*, *Configuration Extrapolation*, *Effort/Cost Estimation*, and *Developer Recommendation.*

Mining Software Repositories. As one of the most popular and widely-used technical Q&A sites in SE communities, **Stack Overflow**[11] **(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. [21] proposed KGXQR, a knowledge graph-based question retrieval approach for programming tasks. KGXQR constructed a software development-related

[11]https://stackoverflow.com/

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. [10] used an inherently interpretable linear regression model [11] to find the configuration with the best predicted performance. They provided interactive visualization charts (e.g., radar charts) to explain the relationships between the application-level configuration parameters and ultimate performance.

Effort/Cost Estimation. Effort/Cost estimation predicts how much effort is required to complete a particular task or project. It is a crucial aspect of project management, playing a significant role in setting realistic timelines and allocating resources efficiently. A representative effort estimation activity is story point estimation, which is a regression task to measure the overall effort required to fully implement a product backlog item. Fu et al. [13] 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. [38] 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.

REFERENCES

[1] Jubril Gbolahan Adigun, Tom Philip Huck, Matteo Camilli, and Michael Felderer. 2023. Risk-driven Online Testing and Test Case Diversity Analysis for ML-enabled Critical Systems. In *Proceedings of the 34th IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 344–354.

[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] Lili Bo, Wangjie Ji, Xiaobing Sun, Ting Zhang, Xiaoxue Wu, and Ying Wei. 2024. ChatBR: Automated assessment and improvement of bug report quality using ChatGPT. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 1472–1483.

[4] Simin Chen, Zexin Li, Wei Yang, and Cong Liu. 2024. DeciX: Explain Deep Learning Based Code Generation Applications. *Proc. ACM Softw. Eng.* 1, FSE (2024), 2424–2446.

[5] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, 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.

[6] 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.

[7] Fabiano Dalpiaz, Davide Dell'Anna, Fatma Basak Aydemir, and Sercan Çevikol. 2019. Requirements Classification with Interpretable Machine Learning and Dependency Parsing. In *Proceedings of the 27th IEEE International Requirements Engineering Conference (RE)*. IEEE, 142–152.

[8] 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.

[9] 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.

[10] 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.

[11] Julian J Faraway. 2004. *Linear Models with R*. Chapman and Hall/CRC.

[12] 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.

[13] Michael Fu and Chakkrit Tantithamthavorn. 2023. GPT2SP: A Transformer-Based Agile Story Point Estimation Approach. *IEEE Trans. Software Eng.* 49, 2 (2023), 611–625.

[14] 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.

[15] Qing Huang, Zishuai Li, Zhenchang Xing, Zhengkang Zuo, Xin Peng, Xiwei Xu, and Qinghua Lu. 2024. Answering Uncertain, Under-Specified API Queries Assisted by Knowledge-Aware Human-AI Dialogue. *IEEE Trans. Software Eng.* 50, 2 (2024), 280–295.

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi 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.

[21] 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.

[22] 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*.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] Ratnadira Widyasari, Gede Artha Azriadi Prana, Stefanus A. Haryono, Yuan Tian, Hafil Noer Zachiary, and David Lo. 2022. XAI4FL: Enhancing Spectrum-Based Fault Localization with Explainable Artificial Intelligence. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC)*. ACM, 499–510.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Jia Yang, Cai Fu, Fengyang Deng, Ming Wen, Xiaowei Guo, and Chuanhao Wan. 2023. Toward Interpretable Graph Tensor Convolution Neural Network for Code Semantics Embedding. *ACM Trans. Softw. Eng. Methodol.* 32, 5 (2023), 115:1–115:40.

[41] 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.

[42] Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. 2024. Chain-of-Event: Interpretable Root Cause Analysis for Microservices through Automatically Learning Weighted Event Causal Graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE)*. ACM, 50–61.

[43] 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.

[44] Hao Yu, Yiling Lou, Ke Sun, Dezhi Ran, Tao Xie, Dan Hao, Ying Li, Ge Li, and Qianxiang Wang. 2022. Automated Assertion Generation via Information Retrieval and Its Integration with Deep learning. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. ACM, 163–174.

[45] He Zhang, Muhammad Ali Babar, and Paolo Tell. 2011. Identifying Relevant Studies in Software Engineering. *Inf. Softw. Technol.* 53, 6 (2011), 625–637.

[46] 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.

[47] 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.