Machine Learning for Actionable Warning Identification: A Comprehensive Survey

XIUTING GE, Nanjing University, Nanjing, China
CHUNRONG FANG[*], Nanjing University, Nanjing, China
XUANYE LI, Nanjing University, Nanjing, China
WEISONG SUN, Nanyang Technological University, Singapore, Singapore
DAOYUAN WU, The Hong Kong University of Science and Technology, Hong Kong, China
JUAN ZHAI, University of Massachusetts Amherst, Amherst, United States
SHANG-WEI LIN, School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore
ZHIHONG ZHAO, Nanjing University, Nanjing, China
YANG LIU, Nanyang Technological University, Singapore, Singapore
ZHENYU CHEN[*], Nanjing University, Nanjing, China

Actionable Warning Identification (AWI) plays a crucial role in improving the usability of static code analyzers. With recent advances in Machine Learning (ML), various approaches have been proposed to incorporate ML techniques into AWI. These ML-based AWI approaches, benefiting from ML's strong ability to learn subtle and previously unseen patterns from historical data, have demonstrated superior performance. However, a comprehensive overview of these approaches is missing, which could hinder researchers and practitioners from understanding the current process and discovering potential for future improvement in the ML-based AWI community. In this paper, we systematically review the state-of-the-art ML-based AWI approaches. First, we employ a meticulous survey methodology and gather 51 primary studies from 2000/01/01 to 2023/09/01. Then, we outline a typical ML-based AWI workflow, including warning dataset preparation, preprocessing, AWI model construction, and evaluation stages. In such a workflow, we categorize ML-based AWI approaches based on the warning output format. Besides, we analyze the key techniques used in each stage, along with their strengths, weaknesses, and distribution. Finally, we provide practical research directions for future ML-based AWI approaches, focusing on aspects like data improvement (e.g., enhancing the warning labeling strategy) and model exploration (e.g., exploring large language models for AWI).

[*] Chunrong Fang and Zhenyu Chen are the corresponding authors.

Authors' Contact Information: Xiuting Ge, Nanjing University, Nanjing, China; e-mail: dg20320002@smail.nju.edu.cn; Chunrong Fang, Nanjing University, Nanjing, China; e-mail: fangchunrong@nju.edu.cn; Xuanye Li, Nanjing University, Nanjing, China; e-mail: 1525135604@qq.com; Weisong Sun, Nanyang Technological University, Singapore, Singapore; e-mail: weisong.sun@ntu.edu.sg; Daoyuan Wu, The Hong Kong University of Science and Technology, Hong Kong, China; e-mail: daoyuan.wu@ntu.edu.sg; Juan Zhai, University of Massachusetts Amherst, Amherst, United States; e-mail: juanzhai@umass.edu; Shang-Wei Lin, School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore; e-mail: shang-wei.lin@ntu.edu.sg; Zhihong Zhao, Nanjing University, Nanjing, China; e-mail: zhaozhih@nju.edu.cn; Yang Liu, Nanyang Technological University, Singapore, Singapore, Singapore; e-mail: yangliu@ntu.edu.sg; Zhenyu Chen, Nanjing University, Nanjing, China; e-mail: zychen@nju.edu.cn.

ACM 1557-7341/2024/9-ART
https://doi.org/10.1145/3696352

CCS Concepts: • **General and reference → Surveys and overviews**; • **Software and its engineering → Software verification and validation**; • **Computing methodologies → Machine learning**.

Additional Key Words and Phrases: Machine learning, static analysis warnings, actionable warning identification, literature review

1 Introduction

Static Code Analyzers (SCAs) can automatically detect defects without executing the program and have been proven to be important and effective in software quality assurance [74]. However, SCAs often generate an overwhelming number of warnings, most of which are unactionable (e.g., false positives) [57, 76, 79]. Statistics show that there are on average 40 warnings per thousand lines of source code [95] and 35%~91% of warnings from SCAs are unactionable [28]. Manually partitioning warnings into actionable and unactionable ones is time-consuming [68] and error-prone [39]. As such, massive unactionable warnings and the cost of manual inspection pose significant obstacles to the practical usage of SCAs [102].

There has been extensive research on improving the usability of SCAs. Various approaches [4] have been proposed to increase the precision of SCAs from the vendor's perspective, thereby minimizing false positives. However, due to the undecidable nature of program behaviors, it is inevitable that SCAs report false positives [84]. Therefore, an alternative approach is *Actionable Warning Identification (AWI)* [24, 28, 66, 67], which is proposed from the user's perspective. These AWI approaches use different techniques (e.g., clustering, ranking, pruning, automated elimination of false positives, static and dynamic combination analysis, or simplifying manual inspection) [67] to postprocess warnings reported by SCAs, thereby classifying or ranking actionable warnings. Machine Learning (ML)-based AWI approaches are notably popular, which train a model with historical warnings and use it to identify actionable warnings on new ones [18, 107]. Due to ML's powerful ability to learn subtle and previously unseen patterns from historical data, ML-based AWI approaches have demonstrated superior performance in enhancing the usability of SCAs [47, 113].

Over the past few years, the substantial progress in the ML-based AWI community has attracted considerable attention from researchers and practitioners. Currently, several existing AWI literature reviews have been proposed to enumerate [28] or categorize [24, 66, 67] specific AWI approaches. Different from the existing reviews that mainly focus on postprocessing techniques of warnings, ML-based AWI approaches require unique characteristics (i.e., the heavy reliance on warning datasets, warning features, and model selection) to identify actionable warnings. However, the existing reviews overlook such unique characteristics, which could present various challenges in developing new and advanced ML-based AWI approaches. For example, warning datasets highly depend on SCAs employing different techniques and projects using various development languages, which could affect the warning feature extraction ways. Also, different warning labeling strategies (e.g., closed warning-based heuristic [102]) could impact the quality of warning datasets and thus affect the ML-based AWI performance. Moreover, different categories of features (e.g., content-based or sequential) necessitate selecting appropriate models for warning representation, leading to varying AWI performance. These diverse design options could hinder researchers and practitioners from further advancements in the ML-based AWI research direction.

To fulfill the above gaps, we are the first to conduct a comprehensive survey by retrospectively examining the current state-of-the-art ML-based AWI studies after years of development. Through analyzing these studies, we first outline a typical ML-based AWI workflow, involving warning dataset preparation, warning dataset preprocessing, AWI model construction, and AWI model evaluation stages. In such a workflow, we categorize ML-based AWI approaches based on the warning output format. Besides, we detail the techniques used in each stage of such a workflow by discussing their strengths and weaknesses and presenting their distribution across different categories of ML-based AWI approaches. Finally, we provide several practical research directions for the

Table 1. Search keywords.

No.	Goal	Keyword
1	machine learning	1) machine learning, 2) deep learning
2	static analysis	1) static analysis, 2) automated code analysis, 3) source code analysis
3	warning	1) warning, 2) alert, 3) alarm, 4) violation
4	identification	1) identifying, 2) elimination, 3) reduction, 4) pruning, 5) classification, 6) prioritization, 7) ranking, 8) reviewing, 9) inspection, 10) simplification

ML-based AWI community. In summary, we believe that our survey can help researchers and practitioners gain a comprehensive understanding and foster progress toward advanced practices in the ML-based AWI field. Our survey makes the following major contributions:

- **Survey methodology.** We employ a meticulous survey methodology across five digital libraries from 2000 to 2023 to gather 51 primary ML-based AWI studies.
- **ML-based AWI.** We outline a typical workflow of applying ML techniques to AWI approaches, which involves warning dataset preparation, warning dataset preprocessing, AWI model construction, and AWI model evaluation.
- **Elaborate study.** We conduct a detailed analysis of the typical ML-based AWI workflow. Such an analysis includes the categorization of ML-based AWI approaches based on the warning output format as well as the discussion of key techniques with associated strengths, weaknesses, and distribution across different categories of ML-based AWI approaches.
- **Practical directions.** We highlight nine practical research directions for future ML-based AWI from the perspectives of data improvement and model exploration.
- **Available artifacts.** We share primary ML-based AWI studies along with associated artifacts in a public repository [108], which facilitates following and extending our survey.

2 Survey Methodology

Guided by the work of Budgen et al. [7], we collect the relevant ML-based AWI studies from the population via a well-designed survey methodology. Such a methodology includes data sources, search keywords, selection criteria, selection procedure of primary studies, and data extraction and synthesis. Further, we perform the trend observations based on the primary studies.

Data sources. We search five popular digital libraries, including (1) IEEE Xplore, (2) ACM, (3) ScienceDirect, (4) Springer Link, and (5) Wiley. These libraries archive various and leading journals and conferences from the software engineering domain [34].

Search keywords. Based on the work of Muske et al. [67], we extend the search keywords to identify the relevant ML-based AWI studies from every digital library. Table 1 shows the search keywords. Such search keywords involve four goals (i.e., ML, static analysis, warning, and identification) with a total of 19 keywords. To search ML-based AWI studies, the four goals are required to be incorporated to create a complete search string. Specifically, for each goal, the logical operator "AND" is used. For each keyword in each goal, the logical operator "OR" is used. With these search strings, we search the above five digital libraries by performing the keyword-based matching in the metadata (i.e., title, abstract, and keywords) of the study [40]. The search scope in each digital library is from 2000/01/01∼2023/09/01. Such a search scope is considered because ML techniques gradually gain attention in the AWI community [28].

Selection criteria. The selection criteria are used to further determine the relevant ML-based AWI studies from the search keywords-based results. Such criteria are initially obtained by steering 10 arbitrarily selected

Table 2. Selection results of primary studies.

Digital library	No. of studies from the search keywords-based results	No. of studies after the selection criteria
IEEE Xplore	85	25
ACM	913	13
ScienceDirect	35	4
Springer Link	70	5
Wiley	150	0
All	1253	47
Removing 15 duplicate studies		32
Snowballing on 32 studies		19
A total of studies		51

studies and are refined based on the pilot search results. Specifically, there are three inclusion criteria, including (1) studies that incorporate ML techniques into AWI; (2) studies that record the contextual items, including datasets, features, ML models, and evaluation details; and (3) studies that locate from 2000/01/01~2023/09/01. There are seven exclusion criteria, including (1) studies that identify warnings via non-ML techniques, e.g., [37, 55, 103]; (2) studies that improve the precision of SCAs, e.g., [96]; (3) studies that evaluate the precision of SCAs, e.g., [74]; (4) studies that track the software quality evolution or SCA rule configuration by investigating warnings, e.g., [1, 95]; (5) studies that identify vulnerable/malicious behaviors via ML techniques and static analysis, e.g., [77]; (6) studies that recommend SCAs to different projects, e.g., [71]; and (7) studies that are not peer-reviewed, e.g., [30]. When both the metadata and full text of a study satisfy inclusion and exclusion criteria, and this study is determined to be relevant.

Selection procedure of primary studies. Table 2 shows the detailed selection results of ML-based AWI studies. Specifically, the search keywords-based results contain 1253 studies. Then, after applying the selection criteria for each study, there involve 47 studies. After that, we rely on the titles of studies to remove 15 duplicate studies and determine 32 distinct studies. However, the search keywords in Table 1 may be incomplete due to the terminological differences among studies. To alleviate this problem, we conduct the snowballing [106], including backward and forward snowballing. Given a good start set, the snowballing can help more effectively and efficiently locate high-quality studies in obscure locations [23]. As such, we take 32 distinct studies as a start set, conduct the snowballing on this start set, and use the selection criteria to search 19 other relevant studies. Finally, we determine 51 primary ML-based AWI studies. Particularly, in the selection procedure of studies, we conduct the 2-pass review. That is, every study goes through two authors (i.e., the first and third authors in our survey). When there is a disagreement on the inclusion or exclusion of a study, the two authors make the discussion and resolve the difference.

Data extraction and synthesis. To ensure data extraction consistency among primary studies, we formulate a unified data extraction form based on a pilot study. To ensure the data extraction correctness in each primary study, the extracted data is checked by two authors. Specifically, the first author of our survey extracts the data from all primary studies. The extracted data is assigned and checked by the third author of our survey. If there is disagreement on the extracted data, the two authors discuss and reach an agreement. In all, 22 data items are extracted for each primary study. Due to the limited space, the detailed data items are shown in a public repository [108]. After that, the extracted data is synthesized via quantitative and qualitative analysis, which can assist us in answering research questions in Section 4.

Trend observations. Fig. 1 shows the statistical distribution of 51 primary studies. The results show that most studies are from the conference, which accounts for 78% (i.e., 40) of 51 primary studies. The remaining studies

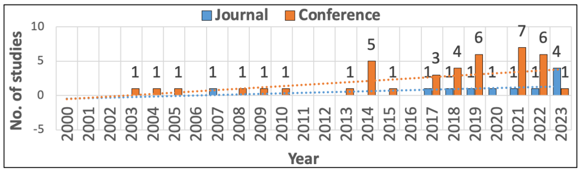


Fig. 1. Distribution of primary studies. The blue (yellow) dashed lines represent the trends in the number of journal (conference) studies over years.

(i.e., 11) are from the journal. Besides, the number of studies by year has steadily increased, which indicates that the ML-based AWI community has received more and more attention from researchers and practitioners in recent years.

3 Background and Related Work

Static analysis warnings. To help developers quickly and easily locate software defects, warnings reported by SCAs generally involve characteristics with the category, severity, message, and location. Of these, the warning location consists of the class and method information containing this warning and the warning line numbers. Due to the limited space, a warning example from SpotBugs is shown in a public repository [108]. Based on whether warnings are ignored by developers, warnings can be divided into actionable and unactionable ones [57, 102, 116]. Such a classification focuses on emphasizing the importance of developer perception [87]. Specifically, an actionable warning is acted on and fixed by developers. By contrast, an unactionable warning is ignored by developers due to many possible reasons (e.g., the over-approximation behaviors [84] and bugs [101, 119] of SCAs). Formally, given a set of commits $C = \{c_1, ..., c_i, ..., c_n\}$ in a project (c_n is the latest commit), a SCA is used to scan the source code of c_i and a set of warnings $W_i = \{w_{i1}, ..., w_{ij}, ..., w_{im}\}$ (m is the number of warnings in c_i) is obtained. If w_{ij} disappears via the warning-related source code change in any commit from c_{i+1} to c_n, w_{ij} is denoted as an actionable warning. If w_{ij} persists from c_{i+1} to c_n, w_{ij} is denoted as an unactionable warning.

AWI. Given a set of warnings reported by a SCA, AWI [67, 87] is to identify actionable warnings from all reported warnings, thereby (1) reducing the number of warnings before reporting them to SCA users; (2) prioritizing warnings that are more likely to be actionable ahead of other warnings; and (3) simplifying manual inspection effort of warnings.

ML-based AWI. ML-based AWI is to extract features from historical warnings, learn a model on the extracted features, and use this model to identify actionable warnings from targeted warnings. ML-based AWI is often formalized into a supervised learning-based problem. Given a set of historical warnings $(X, Y) = \{(x_1, y_1), ..., (x_i, y_i), ..., (x_n, y_n)\}$, (x_i, y_i) $(1 \leq i \leq n)$ is a historical warning. As for $x_i \in X = \{wf_{i,1}, ..., wf_{i,j}, ..., wf_{i,m}\}$, $wf_{i,j}$ $(1 \leq j \leq m)$ is a warning feature. As for $y_i \in Y = \{0, 1\}$, y_i is the warning label of x_i, where $y_i = 0/1$ denotes that x_i is an unactionable/actionable warning respectively. ML-based AWI relies on historical warnings (X, Y) to learn a decision function $Y = f(X)$, aiming to describe the mapping relations between warning features and warning labels. When given a targeted warning X_{target}, the learned decision function is used to predict a corresponding output y_{target}. In general, such an output format could be a binary or continuous value.

Existing literature reviews. Currently, there have been four related AWI literature reviews [24, 28, 66, 67]. Table 3 shows the differences between our survey and the four reviews. *First*, our survey spans a longer period of search scope (i.e., about 23 years from 2000/01/01 to 2023/09/01) compared to the four reviews. Such an enlarged search scope helps our survey increase about 20 new studies that have never appeared in the four reviews, which can facilitate researchers and practitioners tracking and refreshing the current state-of-the-art

Table 3. Comparison of differences between the existing literature reviews and our survey.

Study	Search scope	Approach category	Warning dataset preparation		Warning dataset processing			AWI model construction				AWI model evaluation		Guidelines in ML-based AWI
			Warning dataset acquisition	Warning dataset labeling	Warning feature category	Warning feature selection	Warning dataset rebalancing	Model category in AWI	Learning category in AWI	AWI model structure	AWI construction scenario	Validation strategy in AWI	Performance measure in AWI	
[28]	1998-2009	✓	✗	✗	✓	✗	✗	✓	✗	✗	✗	✓	✓	✗
[67]	2002-2006	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[66]	2002-2020	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[24]	2003-2022	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗
Ours	2000-2023	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

ML-based AWI process. *Second*, our survey conducts a more comprehensive analysis of ML-based AWI. On the one hand, unlike the four reviews that enumerate or categorize AWI approaches based on the postprocessing techniques of warnings, our survey categorizes ML-based AWI approaches based on the warning output format. Such an approach category can help researchers and practitioners gain a deeper understanding of how ML techniques work for AWI. On the other hand, in each stage of the typical ML-based AWI workflow, our survey analyzes the key techniques with associated strengths and weaknesses as well as exhibits the distribution of these key techniques across different categories of ML-based AWI approaches. Such a detailed analysis can provide researchers and practitioners with a thorough understanding of the ML-based AWI field. *Third*, our survey provides more targeted guidelines for the ML-based AWI field. Based on the analysis results, our survey highlights practical guidelines from the perspective of data improvement and model exploration when applying ML techniques in AWI, which can assist researchers and practitioners in enhancing ML-based AWI approaches in a targeted manner. Particularly, the goal between our survey and the review of Guo et al. [24] is different. Our survey focuses on AWI, while the review of Guo et al. only centers on false positive mitigation. As shown in the above warning classification, AWI can embrace a more extensive research scope than false positive mitigation. This indicates that the findings of our survey could be more practical and flexible than those of the review of Guo et al.

4 Detailed Analysis of ML-based AWI

4.1 Typical ML-based AWI Workflow and Research Questions

In this section, we describe a typical ML-based AWI workflow and present the proposed Research Questions (RQs) based on such a workflow.

Typical ML-based AWI workflow. By analyzing the primary studies, we outline a typical ML-based AWI workflow. As shown in Fig. 2, this workflow mainly contains four stages, i.e., warning dataset preparation, warning dataset preprocessing, AWI model construction, and AWI model evaluation. In particular, the first and second stages focus on the data part of ML-based AWI, and the third and fourth stages focus on the model part of ML-based AWI.

(1) In the warning dataset preparation stage, given a set of projects and SCAs, the collected warnings with associated labels are returned. This stage mainly contains warning dataset acquisition and labeling. The warning dataset acquisition collects warnings from SCAs with various static analysis techniques and projects with different development languages. The warning dataset labeling assigns labels for collected warnings. In general, the well-prepared warnings are split into the training and test set.

(2) In the warning dataset preprocessing stage, the well-prepared warnings are preprocessed via different ways. According to the existing ML-based AWI approaches [18, 47, 113, 115], the common ways to preprocess warnings include warning feature extraction, warning feature selection, and warning dataset rebalancing. Specifically, the warning feature extraction mines useful features from the well-prepared warnings. The warning feature selection

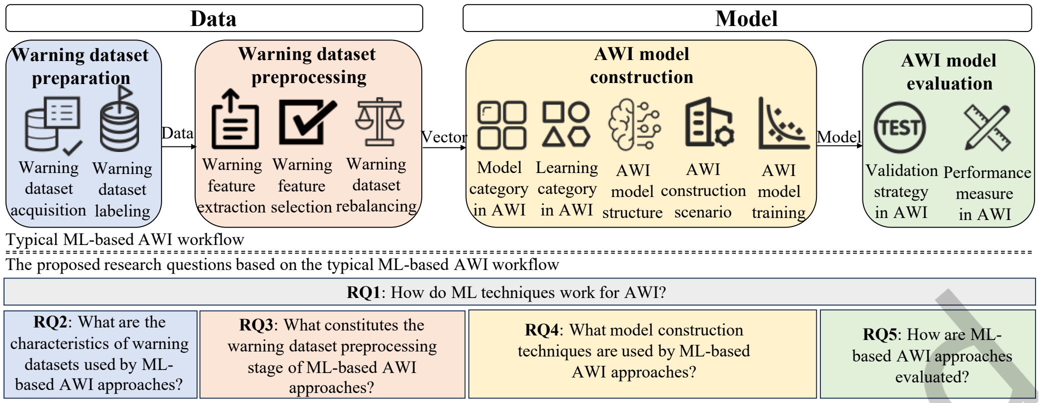

Data
Model
Warning dataset preparation
Warning dataset acquisition
Warning dataset labeling
Data
Warning dataset preprocessing
Warning feature extraction
Warning feature selection
Warning dataset rebalancing
Vector
AWI model construction
Model category in AWI
Learning category in AWI
AWI model structure
AWI construction scenario
AWI model training
Model
AWI model evaluation
Validation strategy in AWI
Performance measure in AWI
Typical ML-based AWI workflow
The proposed research questions based on the typical ML-based AWI workflow
RQ1: How do ML techniques work for AWI?
RQ2: What are the characteristics of warning datasets used by ML-based AWI approaches?
RQ3: What constitutes the warning dataset preprocessing stage of ML-based AWI approaches?
RQ4: What model construction techniques are used by ML-based AWI approaches?
RQ5: How are ML-based AWI approaches evaluated?

Fig. 2. Typical ML-based AWI workflow and the proposed RQs based on such a workflow.

is to select the discriminative warning feature subset from the original warning features. The warning dataset rebalancing is to alleviate the class imbalance in the well-prepared warnings. In general, for ML-based AWI approaches, the warning feature extraction is a necessary step, while the remaining two ways are optional steps.

(3) In the AWI model construction stage, different ML techniques (e.g., Random Forest) are used to train an AWI model based on the training set. Generally, this stage involves model category, learning category, AWI model structure, AWI construction scenario, and AWI model training. The model category (e.g., deep learning) denotes the category of ML techniques used for AWI. The learning category (e.g., supervised learning) represents how an ML-based AWI model learns from the training set. The AWI model structure (e.g., the base structure) denotes how to organize ML techniques to train an AWI model. The AWI construction scenario (e.g., within project) represents the application scenario of an ML-based AWI model. The AWI model training is responsible for constructing an optimal ML-based AWI model from the training set.

(4) In the AWI model evaluation stage, the performance of the well-constructed AWI model is evaluated on the test set by setting the validation strategy and selecting the performance measure. The validation strategy signifies how the well-prepared warnings are split into the training and test sets. The performance measure shows how a well-constructed AWI model performs.

Research questions. Inspired by the typical ML-based AWI workflow, we propose five Research Questions (RQs) to analyze the existing ML-based AWI studies.

- *RQ1: How do ML techniques work for AWI?*
- *RQ2: What are the characteristics of warning datasets used by ML-based AWI approaches?*
- *RQ3: What constitutes the warning dataset preprocessing stage of ML-based AWI approaches?*
- *RQ4: What model construction techniques are used by ML-based AWI approaches?*
- *RQ5: How are ML-based AWI approaches evaluated?*

As described in Fig. 2, RQ1 analyzes the primary studies from a holistic perspective of ML-based AWI, and RQ2 ~ RQ5 analyze the primary studies from an atomistic perspective of ML-based AWI. Specifically, RQ1 aims to classify ML-based AWI approaches based on the warning output format, thereby providing an overall understanding of ML techniques' application in AWI. RQ2 ~ RQ5 correspond with the four stages of such a workflow, which aim to provide insights for the key techniques used in the warning dataset preparation, warning dataset preprocessing, AWI model construction, and AWI model evaluation, respectively.

4.2 RQ1: How do ML techniques work for AWI?

As shown in Section 3, given a targeted warning in the test set, the well-constructed ML-based AWI model in the typical ML-based AWI workflow gives an output for this warning. Such an output format could be a binary value, a continuous value, or both binary and continuous values. Based on the warning output format, ML-based AWI approaches can be divided into classification, ranking, and combination approaches. Further, we illustrate the three categories of approaches along with their strengths and weaknesses, thereby understanding the application of ML techniques in AWI.

Classification approach. The classification approach aims to learn an ML-based AWI classifier based on historical warnings and use this classifier to classify targeted warnings into actionable and unactionable ones. That is, given a targeted warning, the output of this classifier is a binary value, which indicates an actionable or unactionable one. Subsequently, actionable warnings are shown to developers for inspection, while unactionable warnings are pruned. Thus, the classification approach can reduce the number of warnings for manual inspection. However, since the pruned warnings are not guaranteed to be false positives, the classification approach may result in false negatives. In the primary studies, 31 studies [3, 5, 14, 15, 18, 27, 31, 32, 39, 41, 46, 47, 51, 62, 63, 78, 85, 93, 94, 102, 105, 107, 109, 110, 112–114, 117, 118, 122, 123] fall into the classification approach.

Ranking approach. The ranking approach learns an ML-based AWI sorter from historical warnings and uses this sorter to prioritize targeted warnings. Instead of a binary value in the classification approach, this sorter outputs a continuous value for each targeted warning. This continuous value denotes the probability that a targeted warning is actionable, and warnings with higher probabilities are ordered up in the list and can be inspected by developers earlier. Thus, the ranking approach is generally considered a regression problem [42]. As no warnings are pruned, the ranking approach does not cause false negatives. However, the number of warnings is not reduced and all reported warnings are still required manual inspection. In the primary studies, 17 studies [9, 33, 36, 44, 49, 50, 56, 61, 70, 76, 79, 80, 82, 83, 92, 98, 120] fall into the ranking approach.

Combination approach. The combination approach, involving three studies [26, 90, 115], combines classification and ranking approaches to identify targeted warnings. Specifically, Yoon et al. [115] first use the classification approach to classify warnings into actionable and unactionable ones, then prune unactionable warnings, and finally use the ranking approach to prioritize the classified actionable warnings. It indicates that this study inherits the weaknesses of the classification approach (i.e., causing false negatives) and the strengths of the ranking approach (i.e., helping developers inspect earlier warnings that are more likely to be actionable). Two studies [26, 90] give binary and continuous values to classify and prioritize targeted warnings. Due to simultaneously displaying the warning label and warning probability, the two studies can provide more auxiliary information to help developers inspect warnings in comparison to the classification or ranking approach. Due to no warning pruning, the two studies do not result in false negatives, while all reported warnings are still manually inspected.

Summary RQ1: Based on the warning output format, ML-based AWI approaches can be divided into classification, ranking, and combination approaches. Particularly, the classification approach, covering nearly 61% of primary studies, is the most commonly used.

4.3 RQ2: What are the characteristics of warning datasets used by ML-based AWI approaches?

The warning dataset is a basic component in the ML-based AWI approach. Based on the typical ML-based AWI workflow in Fig. 2, the warning dataset preparation mainly involves the warning dataset acquisition and labeling. To reveal the warning dataset characteristics, we analyze warning dataset acquisition and labeling techniques with associated strengths, weaknesses, and distribution across three categories of ML-based AWI approaches, respectively.

Table 4. SCA details, including a SCA name with the correspondingly embedded link/reference, commonly supported languages, the fact whether the project under test is required to be compilable before the usage of a SCA (Comp.), and mapping studies of a SCA.

No.	Name	Languages	Comp.	Studies	No.	Name	Languages	Comp.	Studies
1	FindBugs (16)	Java	Yes	[18, 26, 27, 39, 41, 56, 79, 85, 94, 102, 105, 107, 109, 110, 112, 123]	15	PMD (1)	Java	No	[56]
2	CppCheck (5)	C/C++	No	[14, 70, 82, 83, 98]	16	Jlint (1)	Java	Yes	[56]
3	Sparrow (4)	Java, C/C++	No	[9, 33, 44, 115]	17	Lint4J (1)	Java	Yes	[56]
4	Chord (3)	Java	Yes	[61, 80, 120]	18	DTS[111] (1)	C/C++	Yes	[122]
5	FindSecBugs (3)	Java	Yes	[46, 47, 113]	19	CBMC (1)	C/C++	No	[113]
6	Flawfinders (2)	C/C++	No	[70, 98]	20	JBMC (1)	Java	Yes	[113]
7	RATS (2)	C/C++, PHP, Python, Perl	No	[70, 98]	21	Rosecheckers (1)	C/C++	No	[14]
8	WAP (2)	PHP	No	[62, 76]	22	SonarQube (1)	Java	No	[31]
9	Pixy (2)	PHP	No	[62, 76]	23	SCATE (1)	Java, C/C++	No	[36]
10	Clang (2)	C/C++	No	[82, 83]	24	RIPS (1)	PHP	No	[76]
11	Frama-C (2)	C	No	[82, 83]	25	phpSAFE (1)	PHP	No	[76]
12	Infer (2)	Java, C/C++, Object-C	No	[41, 123]	26	WeVerca (1)	PHP	No	[76]
13	MC[12] (2)	C/C++	No	[49, 50]	27	phpMiner[88] (1)	PHP	No	[62]
14	Airac (2)	C/C++	No	[36, 114]	28	Anonymous (12)	Java, C/C++, Solidity, JavaScript	N/A	[5, 14, 15, 32, 51, 63, 78, 90, 92, 93, 117, 118]

4.3.1 Warning dataset acquisition. The warning dataset acquisition is to collect warnings by using SCAs to automatically scan the source code of a project under test. Thus, the warning dataset acquisition involves the SCA, the development language of the project, and the project source.

SCA. As shown in Table 4, the most commonly used SCA is FindBugs, followed by CppCheck and Sparrow. In all, there are 27 known SCAs mentioned in the primary studies. In particular, 12 studies (e.g., [90, 117]) do not disclose the names of SCAs because these SCAs are only commercially available. Also, some studies (e.g., [56, 82]) use multiple SCAs for the warning dataset acquisition. SCAs detect software defects by adopting various techniques. For example, FindBugs relies on the pattern matching [97] to identify potentially dangerous source code. CppCheck uses the flow-sensitive analysis [43] to reveal undefined program behaviors. Sparrow is designed to detect defects by using the abstract interpretation technique to approximate program behaviors [10]. SCAs with various techniques could yield different results, especially for the warning category and amount. As such, the warning dataset acquisition is greatly dependent on a specific SCA.

Development language of the project. Table 5 shows that most projects focus on Java and C/C++, which account for 49% (25/51) and 45% (23/51) of primary studies respectively. Also, a few projects, involving a total of four studies, are PHP, JavaScript, and Solidity. In particular, the study [113] simultaneously pays attention to Java and C/C++ projects. Combined with Table 4 and Table 5, it is observed that the development languages of projects are closely related to SCAs. It indicates that different development languages of projects are related to the selection of SCAs, thereby affecting the warning dataset acquisition.

Project source. By analyzing the primary studies, the project source contains real-world and synthetic categories. The project source determines the category of the acquired warning dataset. Thus, the warning dataset can be divided into real-world and synthetic categories.

Table 5. Development languages of projects under test with corresponding studies.

Language	Studies
Java (25)	[18, 26, 27, 31, 39, 41, 46, 47, 56, 61, 78–80, 85, 90, 94, 102, 105, 107, 109, 110, 112, 113, 115, 120]
C/C++ (23)	[3, 5, 9, 14, 15, 32, 33, 36, 44, 49–51, 63, 70, 82, 83, 98, 113, 114, 117, 118, 122, 123]
PHP (2)	[62, 76]
JavaScript (1)	[93]
Solidity (1)	[92]

The real-world source refers to the warning dataset collected from the real-world project. In the primary studies, the warning datasets in the majority of primary studies (82%) (i.e., [9, 15, 18, 26, 27, 31–33, 36, 39, 41, 44, 49–51, 56, 61–63, 70, 76, 79, 80, 85, 90, 92–94, 98, 102, 105, 107, 109, 110, 112, 114, 115, 117, 118, 120, 122, 123]) are from the real-world source. Further, as shown in Fig. 3a, 24, 15, 3 studies in the real-world source fall into classification, ranking, and combination approaches, respectively. In particular, it is observed that the warning dataset, collected from 12 open-source Java projects in the study [102], is the most widely used to support ML-based AWI. Currently, nine studies [18, 39, 79, 94, 105, 107, 109, 110, 112] have adopted the warning dataset from the study [102]. In the real-world source, the warning dataset can reflect the realistic distribution, while facing the imbalance problem [18] and missing warnings with uncommon categories.

The synthetic source refers that the warning dataset is collected from the artificially designed projects, including Juliet[1] and OWASP[2]. Juliet and OWASP focus on C/C++ and Java projects, respectively. As shown in Fig. 3a, seven studies use the warning datasets from the synthetic source. Of these, five studies (i.e., [3, 5, 14, 46, 78]) fall into the classification approach, and two studies (i.e., [82, 83]) fall into the ranking approach. In comparison to the warning dataset in the real-world source, the warning dataset in the synthetic source can easily remain balanced and cover warnings with uncommon categories, while not depicting the realistic warning distribution.

As shown in Fig. 3a, two studies [47, 113] separately perform the warning classification on two categories of warning datasets, where one warning dataset is from the synthetic source (i.e., OWASP) and the other warning dataset is from the real-world source. It indicates that the warning datasets in the two studies can inherit the strengths of real-world and synthetic sources.

4.3.2 Warning dataset labeling. The warning dataset labeling is to assign labels for warnings. In the primary studies, the warning labeling involves manual, automatic, and hybrid strategies.

Manual strategy. The manual strategy relies on developers' domain knowledge to label warnings. As shown in Fig. 3b, the manual strategy involves 22 studies, where 12 [15, 32, 41, 51, 62, 63, 85, 93, 114, 117, 118, 122], 8 [9, 36, 44, 49, 50, 80, 120], and 2 [90, 115] studies fall into classification, ranking, and combination approaches respectively. The manual strategy can obtain a small number of valuable warnings. However, the manual strategy has intrinsic limitations. On the one hand, as illustrated in a study [28], it takes an experienced developer five minutes to inspect each warning on average. It indicates that the manual warning labeling strategy is very time-consuming. Consequently, it is difficult for the manual strategy to quickly gather massive warnings. On the other hand, developers have different levels of experience [21]. Given the same warning inspected by different developers, the label of this warning may be inconsistent. That is, it is error-prone that a warning is manually inspected by only a developer [39].

Automatic strategy. The automatic strategy performs the warning labeling without any manual intervention. Fig. 3b presents that most studies adopt the automatic strategy for warning labeling. Specifically, 14 [3, 5, 14, 18, 27, 31, 46, 78, 94, 102, 105, 109, 110, 123], 6 [56, 61, 79, 82, 83, 92], and 1 [26] studies fall into classification,

[1] https://samate.nist.gov/SARD/test-suites/112

[2] https://owasp.org/www-project-benchmark/

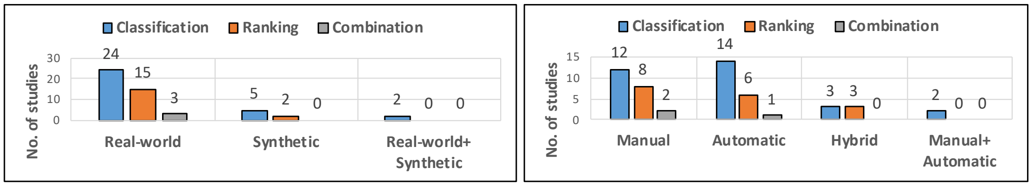
ranking, and combination approaches respectively. Based on different core techniques, the automatic strategy can be further refined into four sub-categories.

First, the warning labels are assigned by judging whether a warning hits the defect-inducing source code. This sub-category is the most commonly used to label warnings in the synthetic source [3, 5, 14, 46, 78, 82, 83], because the source code in the synthetic warning dataset has the oracle. That is, if a warning reported by the SCA hits the defect-inducing source code, this warning is labeled to be actionable. Otherwise, this warning is labeled to be unactionable. In particular, Liang et al. [56] use this sub-category to label warnings in the real-world source. However, there is no well-prepared defect-inducing source code in the real-world source. To address this problem, Liang et al. increase a preliminary step. Specifically, it is observed that SCAs tend to detect generic defects rather than project-specific defects. Based on such an observation, the file modification times are used to select generic defect-inducting revisions. Subsequently, the diff-based algorithm [35] is used to identify the defect-inducing source code from these generic revisions. It is noted that different studies use different granularities (i.e., statement, method, or file levels) to determine whether a warning hits the defect-inducing source code. For example, Alikhashashneh et al. [3] perform the warning labeling at the method level. That is, a warning is determined to be actionable once this warning falls into the method containing the detect-inducing source code. More strictly, in the work of Liang et al. [56], a warning is considered to be actionable only if (1) the source code lines of this warning hit at least one defect-inducing source code line and (2) this warning in the current revision disappears in a later revision. The granularity from file to statement levels generally becomes more fine-grained, and the obtained warning labels are more reliable.

Second, the warning labels are obtained by the closed warning-based heuristic, which involves 11 studies [18, 26, 27, 31, 79, 94, 102, 105, 109, 110, 123]. Inspired by the fact that developers are constantly fixing defects in the program, such a heuristic is to give warning labels by performing the warning matching among revisions [102]. Specifically, to judge whether two given warnings are identical in different revisions, the warning matching is performed by comparing the warning characteristics (e.g., category and location). Then, there are three cases: (1) if a warning in the current revision disappears in any later revision, this warning is considered to be fixed by developers and is labeled to be actionable; (2) if a warning in the current revision is present until the latest revision, this warning is considered to be ignored by developers and is labeled to be unactionable; and (3) if the class/method, where a warning is in the current revision, is deleted in a certain later revision, this warning is labeled to be unknown. However, the current warning matching is severely affected by the warning-irrelevant source code changes (e.g., the class/method renaming or the code refactoring), thereby making the heuristic produce many mislabeled warnings [53, 57, 116].

Third, the warning labels are obtained by the voting mechanism. Inspired by an observation that a defect is identified exclusively by a single SCA [11, 22], Tran et al. [92] perform the warning labeling based on the results reported by multiple SCAs. Specifically, if a warning is only reported by one SCA, this warning is labeled to be unactionable. If a warning is reported by at least two SCAs, this warning is labeled to be actionable. The reliability of a warning being labeled as actionable/unactionable one could be increased due to aggregating the defect detection capabilities of multiple SCAs. However, the voting mechanism-based warning labeling strategy could bring mislabeled warnings. On the one hand, due to the similar or incorrect warning patterns adopted by SCAs, a warning simultaneously reported by multiple SCAs could be unactionable. On the other hand, due to the complementarity of techniques adopted by SCAs [54], a warning that is reported by only one SCA may be actionable. In addition, compared to the number of warnings reported by only one SCA, the voting mechanism-based warning labeling strategy could report more unactionable warnings because warnings are merged from multiple SCAs [6].

Fourth, the warning labels are obtained via the *k*-object-sensitive version [64], which mainly consists of the object sensitivity and parameterization framework. The object sensitivity is a form of context sensitivity for flow-insensitive points-to analysis. The core idea behind object sensitivity is to separately analyze a method



(a) Project sources.

(b) Warning dataset labeling strategies.

Fig. 3. Distribution of the warning dataset preparation under three categories of ML-based AWI approaches.

for each object name, which represents a runtime object that may invoke this method. The parameterization framework, controlled by k, is designed for the tradeoff between cost and precision in the object sensitivity. For example, Mangal et al. [61] adopt the k-object-sensitive version ($k = 4$) to assign labels for warnings reported by a context-sensitive but object-insensitive SCA called Chord. By leveraging the technique complementarity between Chord and the k-object-sensitive version, the warning labels can be easily obtained. However, due to comprising approximations (e.g., flow-insensitivity) in the k-object-sensitive version, these warning labels do not have the absolute ground truth.

Hybrid strategy. The hybrid strategy combines manual and automatic strategies to perform warning labeling. As shown in Fig. 3b, the hybrid strategy involves six studies [39, 70, 76, 98, 107, 112], which fall evenly into classification and ranking approaches. Specifically, two studies [70, 98] manually inspect whether a method is vulnerable and automatically label warnings by judging whether warnings hit vulnerable methods. Yet, the obtained warning labels could be noisy because a vulnerability could be caused by the interprocedural method [70, 98]. Similarly, Pereira et al. [76] adopt the warning dataset in the work of Nunes et al. [72], which automatically extract the Proof-of-Concept vulnerabilities from CVE[3] and manually label warnings by comparing the source code lines of warnings and the vulnerable source code lines. However, the source code lines between warnings and vulnerabilities do not correspond perfectly, and there could be mislabeled warnings [76]. In addition, to alleviate mislabeled warnings caused by the closed warning-based heuristic, Kang et al. [39] rely on the 2-pass manual inspection to further ensure the reliability of warning labels. Subsequently, the warning dataset collected by Kang et al. [39] is also adopted by other studies [107, 112]. The warning labels in Kang et al. [39] can be considered to be ground-truth. However, these ground-truth warnings only occupy a small part of warnings reported by the SCA because the 2-pass manual inspection resource is limited. In summary, the hybrid strategy can to some extent mitigate the weaknesses of manual and automatic strategies. However, it is still a tough task how to combine manual and automatic strategies to improve the warning label quality.

Fig. 3b shows that two studies [47, 113] separately use manual and automatic strategies to label warnings. Different from the hybrid strategy that simultaneously uses manual and automatic strategies to label warnings on the same warning dataset, the two studies use manual and automatic strategies to label warnings on different warning datasets, respectively. Specifically, the manual strategy is used to perform the warning labeling in the real-world source, and the automatic strategy is used to perform the warning labeling in the synthetic source.

[3] https://cve.mitre.org/

Summary RQ2: The warning dataset preparation stage in the ML-based AWI approaches contains warning dataset acquisition and labeling. In the warning dataset acquisition, it is observed that (1) FindBugs, the most commonly adopted SCA, is used as a research target in 31% of primary studies; (2) Java and C/C++ are the primarily focused development languages of projects; and (3) the warning dataset from the real-world source occupies 82% of primary studies. In the warning dataset labeling, the manual and automatic strategies are the most commonly adopted to label warnings, which cover 43% and 41% of primary studies, respectively.

4.4 RQ3: What constitutes the warning dataset preprocessing stage of ML-based AWI approaches?

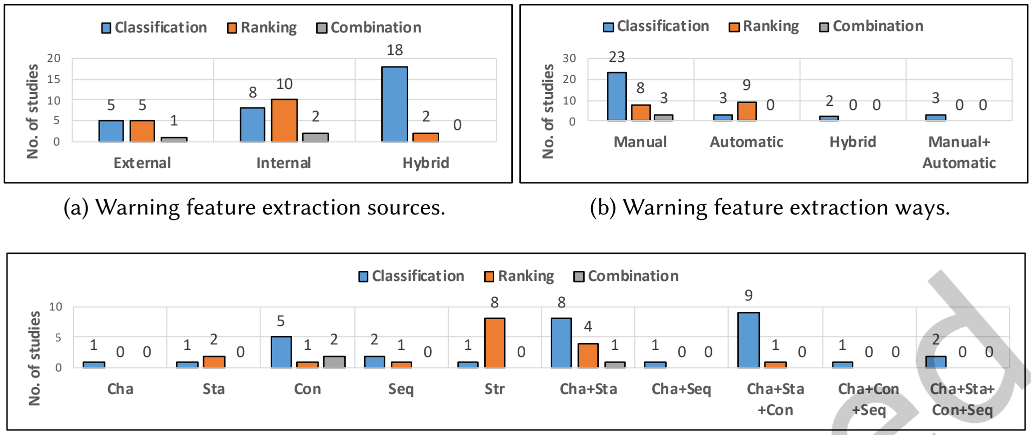
The warning dataset preprocessing produces the core input for ML-based AWI. Based on the typical ML-based AWI workflow in Fig. 2, the warning dataset preparation mainly involves warning feature extraction, warning feature selection, and warning dataset rebalancing. To illuminate the warning dataset preprocessing, we analyze the techniques used in the three parts, along with strengths, weaknesses, and distribution across three categories of ML-based AWI approaches.

4.4.1 Warning feature extraction. This section discusses the warning feature extraction from three perspectives, including extraction sources, extraction ways, and categories of warning features.

Warning feature extraction source. Inspired by the work of Wang et al. [100], the warning feature extraction in primary studies can be divided into external, internal, and hybrid sources.

The external source is to extract warning features from the SCA report or developer feedback. Fig. 4a shows that the external source has 11 studies, where 5 [5, 93, 117, 118, 123], 5 [49, 50, 76, 82, 83], and 1 [90] studies involve classification, ranking, and combination approaches respectively. The warning reported by SCAs contains a series of warning characteristics (e.g., category, severity, and location). Many studies (e.g., [93]) extract the original warning characteristics from the SCA report as warning features. Also, some studies (e.g., [76, 118]) derive some new warning features (e.g., the number of warnings in a class/method) from the original warning characteristics. In addition, a few studies [102, 117] extract warning features from the developer feedback on warnings reported by the SCA. The warning features in the external source are obtained by only parsing the SCA report or the developer feedback. Thus, the warning features are generally easily extracted. However, due to independent of the warning-related source code, the warning features in the external source only capture the shallow warning information and fail to reveal the root cause of the warning.

The internal source is to extract warning features from the source code containing warnings. Fig. 4a shows that the internal source occupies 39% (20/51) of primary studies, where 8 [3, 32, 46, 51, 62, 63, 78, 114], 10 [9, 33, 36, 44, 61, 70, 80, 92, 98, 120], and 2 [26, 115] studies fall into classification, ranking, and combination approaches respectively. Given a warning reported by the SCA, the warning location (i.e., the class and method information containing this warning and warning line numbers) are generally displayed. Based on the warning location, the source code related to the class, method, and warning line numbers can be obtained. Besides, by using program analysis techniques (e.g., program slicing [104]), the source code that traces the path leading to the warning is available. Moreover, based on the software evolution history, the warning-related software change history can be recorded in terms of class, method, and warning lines. For example, Lee et al. [51] extract features from the source code surrounding warning line numbers. Some studies [70, 98] use the abstract syntax tree with control flow construction technique to extract features from the source code containing warnings. Compared to warning features in the external source, the warning features in the internal source could capture deeper information. However, there are some limitations in the internal warning feature extraction source. On the one hand, the warning features in the internal source are more difficult to be extracted than those of the external source. On the other hand, the source code indeed contains rich syntactic and structural information of the warning, but not all of the source code in the project is related to the warning. Thus, to capture comprehensive



(a) Warning feature extraction sources.

(b) Warning feature extraction ways.

(c) Warning feature categories. Cha, Sta, Con, Seq, and Str are characteristic-based, statistical, content-based, sequential, and structural categories, respectively. Studies that compare the AWI performance differences of multiple warning feature categories are merged together (e.g., Cha+Sta).

Fig. 4. Distribution of the warning feature extraction under three categories of ML-based AWI approaches.

warning information while eliminating warning-irrelevant information, it is tough how to appropriately extract warning features from the internal source.

The hybrid source is to extract warning features from external and internal sources. Fig. 4a presents 20 studies in the hybrid source, where 18 [14, 15, 18, 27, 31, 39, 41, 47, 85, 94, 102, 105, 107, 109, 110, 112, 113, 122] and 2 [56, 79] studies belong to classification and ranking approaches respectively. For example, Hegedűs et al. [31] combine the warning category extracted from the SCA report and the source code surrounding warning line numbers as warning features. Wang et al. [102] extract eight categories of warning features (e.g., warning characteristics and software change history) from the SCA report and source code containing the warning. The hybrid source can represent warnings comprehensively by combining warning features with external and internal sources. However, the existing studies directly concatenate warning features from external and internal sources together. Actually, there is a redundancy in the warning features in external and internal sources [107], which severely hinders the AWI performance. Thus, it is investigated how to reasonably combine warning features from external and internal sources.

Warning feature extraction way. Based on whether there is manual intervention [86], the warning feature extraction involves manual, automatic, and hybrid ways.

The manual way is to rely on experts' domain knowledge for warning feature extraction. Fig. 4b shows that such a way involves the most studies, where 23 [3, 5, 14, 15, 18, 27, 32, 39, 62, 85, 93, 94, 102, 105, 107, 109, 110, 112, 114, 117, 118, 122, 123], 8 [36, 49, 50, 56, 76, 79, 82, 83], and 3 [26, 90, 115] studies fall into classification, ranking, and combination approaches respectively. By the manual way, a small number of meaningful features can be obtained and be suitable for homogeneous warnings. For example, Zheng et al. [123] assume that the complex source code (e.g., more condition statements and lines of code) is more likely to be found bugs by the SCA. Based on this assumption, they manually extract 25 features (e.g., frequency of OR/AND conditions in the warning trace and length of warning line numbers) to denote the complexity of the warning-related source code for AWI. However, it is tedious to manually extract warning features. Besides, different experts could yield heterogeneous domain knowledge, thereby causing biased warning features. More fatally, the manually extracted warning features become obsolete over time, which makes it difficult to identify newly reported warnings due to the concept drift [17].

The automatic way is to leverage Deep Learning (DL) or datalog reasoning techniques for warning feature extraction. Fig. 4b shows 12 studies in the automatic way, where 3 [46, 51, 78] and 9 [9, 33, 44, 61, 70, 80, 92, 98, 120] studies cover classification and ranking approaches respectively. Specifically, six studies [46, 51, 70, 78, 92, 98] represent warnings by using DL techniques (e.g., Word2Vec) to encode the warning-related source code. The remaining six studies [9, 33, 44, 61, 80, 120] extract the datalog derivation graph from the warning-related source code via def-use analysis [73] and perform queries for warnings by applying the Bayesian inference algorithm to such a graph. The DL technique can automatically capture the deep warning information. However, the warning features obtained by DL could be sparse and high-dimension as well as require a lot of computing power and massive historical warnings. By contrast, despite maintaining the declarative semantics and logical consistency based on the flexible rule definition in the def-use analysis, the datalog reasoning technique suffers from limited expressiveness in the program with the complex data types and the high computational overhead [65].

The hybrid way is to extract features by combining the manual and automatic ways. Fig. 4b signifies that two studies [31, 63] related to the classification approach use the hybrid way for warning feature extraction. Specifically, Hegedűs et al. [31] manually extract warning characteristics and automatically extract the source code surrounding the warning line numbers, thereby concatenating warning characteristics and source code as warning features. Meng et al. [63] first automatically construct the code property graph from the warning-related source code and manually extract warning features from this code property graph. Compared to warning features in the manual way, the warning features in the hybrid way could be more robust for AWI due to embracing the automatic extraction of warning features. Compared to the warning features in the automatic way, the warning features in the hybrid way are require additional domain knowledge due to relying on the manual way to extract warning features.

Fig. 4b signifies that three studies [41, 47, 113] related to the warning classification separately extract warning features in manual and automatic ways. For example, Kharkar et al. [41] manually extract identifiers related to the null dereference and resource leak (e.g., whether *return null* exists) from the local and non-local warning-related source code as warning features for AWI. Additionally, they automatically extract the local and non-local warning-related source code as warning features and apply the DL model to encode these warning features for AWI. Subsequently, they compare the AWI performance difference under warning features between manual and automatic ways.

Warning feature category. By analyzing the primary studies, the warning features can be classified into characteristic-based, statistical, content-based, sequential, and structural categories. Table 6 shows the details of different warning feature categories.

The characteristic-based category denotes the warning characteristics extracted from the external source, especially the SCA report. The warning features (e.g., category and severity) in the characteristic-based category are commonly used by prior studies (e.g., [5, 31, 117]). It is noted that some SCAs (e.g., Infer) report warnings with associated source code. As such, the warning features (e.g., the source and sink identifiers) extracted from such the source code [93, 123] are also considered as the characteristic-based category. In addition, a few studies [102, 117] extract the developer idea (i.e., the developer feedback on warnings reported by the SCA) as the characteristic-based warning feature. As shown in Fig. 4c, the characteristic-based category involves 28 studies, where one study [5] only uses the characteristic-based warning features for AWI. The characteristic-based warning features are easily extracted. However, due to missing a deep understanding of the warning-related source code, the characteristic-based warning features could lack sufficiently discriminative capability when separately used for AWI.

The statistical category represents the warning statistics information from the SCA report or the source code containing warnings. The warning features from the SCA report (e.g., the number of warnings in a class/method)

Table 6. Warning feature categories with their associated descriptions, strengths, and weaknesses.

Category	Description	Strengths	Weaknesses
Characteristic	This category denotes the warning characteristics extracted from the external source, i.e., the SCA report and developer feedback on the reported warnings.	Being easily extracted	Lacking sufficiently discriminative capability due to missing a deep understanding of the warning-related source code.
Statistical	This category denotes the warning statistics information from two aspects. Features (e.g., the number of warnings in a class) from the SCA report provide aggregated warning information. Features (e.g., the depth and modification times of a class) from the source code containing warnings provide complexity and evolution information.	Unveiling the hidden mathematical distribution of warnings, which is instructive to identify warnings when there are sufficient history warnings	Requiring intensive computation power; Being likely to having the data leak in some warning features due to the implementation mistakes
Content	This category signifies important identifiers in the source code containing warnings, e.g., identifiers related to loops. Unlike the characteristic-based category that focuses on the SCA report, the content-based category focuses on the source code containing warnings.	Capturing warnings with regular patterns due to searching important identifiers from the source code containing warnings	Heavily relying on domain knowledge and being labor-intensive
Sequential	This category denotes the sequential information of the source code containing warnings by considering the source code containing warnings as the natural language text and using DL techniques to encode such the source code. Unlike the content-based category that extracts the important identifiers from the source code containing warnings, the sequential category aims to learn the sequential distribution from the source code containing warnings.	Capturing the sequential information of warnings	Missing the structural information of warnings; Being generally sparse and high-dimension; Being difficult to interpret
Structural	This category embodies the structural information of warnings by mainly extracting warning features from the abstract syntax tree, control flow, or data flow of warnings. Unlike the sequential category that considers the source code containing warnings as the natural language text, the structural category considers the source code containing warnings as the tree/graph structure.	Capturing richer information with the warning syntax and semantics	Being difficult to interpret when DL techniques are used for AWI; Enduring limited expressiveness when the datalog derivation graph is used for AWI

denote the warning aggregation information [27, 102]. The warning features from the source code containing warnings (e.g., the depth and modification times of a method containing warnings) denote the warning complexity and evolution information [56]. As shown in Fig. 4c, the statistical category involves 28 studies, where three studies [3, 49, 50] separately adopt the statistical warning features for AWI. The statistical warning features unveil the hidden mathematical distribution of warnings. When there are sufficient history warnings, it is very instructive to identify newly reported warnings [29, 45, 89, 123]. However, the extraction process of the statistical warning features requires intensive computation power. Also, it is worthy noting that due to the incorrect or inappropriate implementation ways, some statistical warning features (e.g., the defect density for warning type) yield data leakage [39], which could cause exaggeration of the ML-based AWI performance.

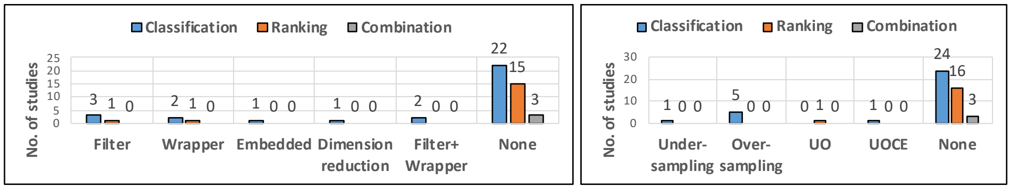
The content-based category signifies important identifiers in the source code containing warnings. The main process to extract the content-based warning features is shown as follows. First, the source code containing warnings is obtained by establishing the abstract syntax tree [26, 113, 115, 122], code property graph [63], program dependency graph [47, 113], local/non-local context [41], or loops-related context [32]. Second, the important identifiers (e.g., identifiers related to loops and library calls [32]) are extracted from the obtained source code. In general, the first step is an automatic process, and the second step is a manually designed process. In particular, two studies [47, 113] automatically extract the content-based warning features for AWI, i.e., applying the Bag-of-Word model [121] to automatically calculate the frequency or occurrence of each identifier in the source code containing warnings. Noted, the difference between the characteristic-based and content-based categories is that the former mainly focuses on the SCA report, and the latter focuses on the source code containing warnings. Fig. 4c presents that 21 studies fall into the content-based category. Of these, eight studies only adopt the content-based warning features for AWI, where five [32, 62, 63, 114, 122], one [36], and two [26, 115] studies are related to classification, ranking, and combination approaches respectively. The content-based warning features can capture warnings with regular patterns due to searching important identifiers from the source code containing warnings, while heavily relying on domain knowledge and being labor-intensive.

The sequential category denotes the sequential information of the source code containing warnings. The sequential warning features are extracted in two steps. First, the source code containing warnings is obtained by performing the program slicing [46, 47, 113] or extracting the source code surrounding the warning line numbers [31, 51, 92]. Second, such source code is considered as the natural language text and is encoded into vectors via DL techniques. Noted, the difference between the content-based and sequential categories is that the former aims to extract the important identifiers from the source code containing warnings, and the latter aims to learn the sequential distribution from the source code containing warnings. Fig. 4c describes that there are seven studies in the sequential category, where three studies [46, 51, 92] separately use the sequential warning features for AWI. The sequential warning features can capture the sequential information of warnings while missing the structural information of warnings. Also, the sequential warning features extracted by DL techniques are generally sparse and high-dimension. Moreover, it is difficult to interpret the role of these warning features in AWI due to the inherent limitations of DL in terms of explainability [8].

The structural category embodies the structural information of warnings by mainly extracting warning features from the abstract syntax tree, control flow, or data flow of warnings. Fig. 4c shows that one study [78] falls into the classification approach and eight studies [9, 33, 44, 61, 70, 80, 98, 120] belong to the ranking approach. Specifically, three studies [70, 78, 98] use DL techniques to automatically learn the structural distribution information from the program dependency graph and control flow-based abstract syntax tree of warnings. Six studies [9, 33, 44, 61, 80, 120] automatically reason the root cause of warnings by using def-use analysis [73] to extract the datalog derivation graph. Noted, the difference between the sequential and structural categories is that the former considers the source code containing warnings as the natural language text, and the latter considers the source code containing warnings as the tree/graph structure. As such, compared to the sequential warning features, the structural warning features can capture richer information with the warning syntax and semantics. However, in the studies [70, 78, 98] that rely on DL techniques to represent warnings for AWI, the structural warning features face the same limitation as those of the sequential category, i.e., difficult to explain [8]. In the studies [9, 33, 44, 61, 80, 120] that adopt the datalog reasoning technique to represent warnings for AWI, the structural warning features endure limited expressiveness when encountering the program with the complex data types [65].

4.4.2 Warning feature selection. Feature selection is an important but optional process in the ML-based tasks [86]. By analyzing the primary studies, the warning feature selection can be classified into filter, wrapper, embedded, and dimension reduction techniques. The filter technique independently ranks features by using statistical metrics without involving any ML model. The wrapper technique measures feature subsets by training and testing a specific ML model and uses its performance as the evaluation metric. The embedded technique determines the best features by integrating feature selection into the model training process. Different from the above three techniques that aim to select a subset of the original features, the dimension reduction technique reduces the number of features by transforming the feature space.

Filter technique. The filter technique selects the warning feature subset by calculating the feature importance or correlation via a certain criterion. Further, based on whether warning labels are required, the warning feature selection involves unsupervised and supervised ways [107]. As shown in Fig. 5a, four studies [3, 76, 94, 107] attempt the filter technique for warning feature selection. For example, Pereira et al. [76] separately use the correlation- and variance-based feature selection techniques for AWI, where the correlation-based one is supervised and the variance-based one is unsupervised. In particular, UNEASE [107], the first unsupervised warning feature selection technique for AWI, mainly performs the feature clustering and feature ranking to select a warning feature subset from the original warning feature set. The filter technique is computationally fast, simple to implement, and model-independent. However, the determination of the warning feature subset in the filter technique often requires manually setting thresholds. Also, due to model independence, the filter technique could ignore model biases and cause low AWI performance.



(a) Warning feature selection techniques.

(b) Warning dataset rebalancing techniques.

Fig. 5. Distribution of the warning feature selection and warning dataset rebalancing techniques under three categories of ML-based AWI approaches. In Fig. 5b, “UO” is to use undersampling and oversampling for AWI. “UOCE” is to use four sampling techniques for AWI.

Wrapper technique. The wrapper technique selects the best warning features by evaluating the ML-based AWI model performance. Fig. 5a shows the wrapper technique in three studies, where two [102, 114] and one [79] studies fall into classification and ranking approaches, respectively. For example, Wang et al. [102] use the greedy backward elimination algorithm for feature selection. Such an algorithm iteratively trains an ML-based AWI classifier, greedily removes features one by one from the original warning features to maximize the performance of this classifier, and finally returns the warning feature subset with the best AWI performance. The wrapper technique can consider dependencies among warning features [86], as it selects the warning feature subset by training and testing an ML-based AWI model, which inherently reflects the interaction among warning features. Thus, the obtained warning feature subset could achieve good AWI performance. However, compared to the filter technique, the wrapper technique is model-dependent and computationally intensive due to the requirement to train an AWI model multiple times.

Embedded technique. The embedded technique automatically performs the warning feature selection when constructing the ML-based AWI model. The embedded technique only involves one study [123] with the classification approach. Specifically, to explain the role of 25 extracted features in AWI, Zheng et al. [123] calculate the warning feature importance via Random Forest (RF). Once RF is well-constructed for AWI, the warning feature importance is obtained. Finally, the root node in RF (e.g., the line number of the error) is dominant for AWI. The embedded technique can select optimal features in parallel to training the ML-based AWI model. Thus, the embedded technique mitigates the weaknesses of the filter and wrapper techniques. However, the embedded technique is model-specific and requires an understanding of the model details.

Dimension reduction technique. The dimension reduction technique maps the original warning feature space to a low-dimensional space via projection. The dimension reduction technique only contains one study [109], which leverages the fractal-based method [52] to convert the high-dimension warning features into a more compressed space without the principal information loss. Compared to the typical dimension reduction technique (i.e., Principal Component Analysis, PCA), Yang et al. [109] consider that the fractal-based method is more appropriate for AWI due to (1) better handling the increasingly sophisticated and non-linearly decomposable data and (2) no need for manually setting thresholds. Besides, the dimension reduction technique is computationally fast because it is model-independent and unsupervised. However, the warning features determined by the dimension reduction technique are difficult to interpret.

Fig. 5a shows that 11 studies adopt the warning feature selection techniques for AWI, of which two studies [27, 117] separately use filter and wrapper techniques for warning feature selection. However, the remaining 40 studies do not adopt any warning feature selection technique for AWI.

4.4.3 Warning dataset rebalancing. Class imbalance, where the number of positive (i.e., actionable) samples is much fewer than that of negative (i.e., unactionable) samples, is a common phenomenon in the warning dataset used for the ML-based AWI model training [18]. To mitigate the above phenomenon, some primary studies adopt the warning dataset rebalancing technique for AWI. By analyzing these primary studies, the warning dataset rebalancing technique involves undersampling, oversampling, combined sampling, and ensemble sampling.

Undersampling technique. The undersampling technique mitigates the class imbalance by eliminating unactionable warnings from the original warning dataset. Fig. 5b describes that only one study [110] related to the classification approach adopts the aggressive undersampling to rebalance the warning dataset. Different from the random undersampling [18, 76], the aggressive undersampling throws away unactionable warnings close to the decision boundary of the ML-based AWI model and accesses actionable warnings until the ratio of actionable and unactionable warnings is balanced in the training set. The undersampling can easily and quickly obtain the balanced classes. However, the undersampling could miss potentially valuable information due to throwing away the majority of unactionable warnings.

Oversampling technique. The oversampling creates a superset of the original warning dataset by generating actionable warnings. As shown in Fig. 5b, five studies [3, 31, 62, 105, 112] use the oversampling to rebalance the warning dataset when performing the warning classification. The commonly used oversampling technique is the Synthetic Minority Oversampling technique (SMOTE), which involves four warning classification studies [3, 62, 105, 112]. Similar to the undersampling, the oversampling can also easily and quickly rebalance the warning dataset. However, the oversampling increases the likelihood of overfitting in the ML-based AWI model [16].

Combined sampling technique. The combined sampling is constructed by combining oversampling and undersampling. Only one study [18] attempts two commonly used combined sampling techniques (i.e., EditNearestNeighbors and SMOTEENN) to perform the warning rebalancing for AWI. The combined sampling can mitigate the weaknesses of undersampling and oversampling techniques. However, compared to the undersampling and oversampling, the combined sampling is more complex and time-consuming.

Ensemble sampling technique. The ensemble sampling solves the class imbalance by embedding the aforementioned sampling techniques into the ensemble learning models. The commonly used ensemble sampling is EasyEnsemble, RUSBoost, BalancedBagging, and BalancedRandomForest [18]. In general, ensemble sampling can help achieve a high AWI performance. However, the ensemble sampling is model-dependent and requires understanding the model details.

As shown in Fig. 5b, two studies [18, 76] attempt different sampling techniques for the warning dataset rebalancing. Specifically, Pereira et al. [76] separately use undersampling and oversampling to mitigate the class imbalance, thereby performing the warning ranking. Ge et al. [76] evaluate the impact of the above four sampling techniques on the ML-based AWI performance. Additionally, 43 studies do not adopt any warning dataset rebalancing technique for AWI.

Summary RQ3: The warning dataset preprocessing stage in the ML-based AWI approaches contains warning feature extraction, warning feature selection, and warning dataset rebalancing. Specifically, it is observed in the warning feature extraction that (1) the main sources are internal and hybrid ones; (2) the manual way is the most prevalent, which occupies 67% of primary studies; (3) the most commonly used warning features fall into the characteristic-based and statistical categories. Particularly, 53% of primary studies combine multiple categories of warning features for AWI. In addition, only 22% and 16% of primary studies attempt the warning feature selection and warning dataset rebalancing techniques for AWI, respectively.

4.5 RQ4: What model construction techniques are used by ML-based AWI approaches?

The ML-based AWI model construction is the key step to provide the decision support for a newly reported warning. By following the typical ML-based AWI workflow in Fig. 2, we present the answer to this RQ by enlisting the techniques used in the model category, learning category, AWI model structure, and AWI construction scenario, along with associated strengths, weaknesses, and distribution across three categories of ML-based AWI approaches.

4.5.1 Model category in AWI. The ML categories used for AWI can be divided into Traditional ML (TML), Deep Learning (DL), statistical inference, and Pre-Trained Model (PTM).

TML model. Fig. 6a shows that 26 studies only use TML models for AWI. Table 7 describes that RF (22/51) are the most commonly used ML models, followed by DT and SVM.

DL model. Table 7 shows eight DL models used for AWI, where CNN occupies the largest proportion. Fig. 6a shows that four studies only use DL models for AWI, where one [51] and three [70, 92, 98] studies are related to classification and ranking approaches, respectively.

Statistical inference model. Fig. 6a shows that the statistical inference model is adopted by nine studies. Specifically, six studies [9, 33, 44, 49, 61, 80] convert the datalog derivation graph of warnings into a Bayesian inference model and utilize the real-time feedback of developers to adjust this model for AWI. Unlike the six studies, Jung et al. [36] directly construct a Bayesian inference model for AWI without any real-time adjustment. In addition, Kremenek et al. [50] rank warnings by applying z-test statistics to the number of historical warnings. Zhang et al. [120] rely on MLN to convert the datalog derivation graph of warnings into a warning prioritization model for AWI.

PTM. PTM performs the self-supervised training on large-scale and unlabeled corpora and then is customized for downstream tasks by fine-tuning on a limited number of labeled samples [25]. Unlike TML and DL models that construct an AWI model from scratch, PTM can be directly used as a starting point. Fig. 6a shows that two studies [41, 112] attempt PTMs for AWI. Specifically, Kharkar et al. [41] pre-train CodeBERTa [59] on a newly collected warning dataset and use a well-pre-trained CodeBERTa to classify targeted warnings. Also, they generate the warning-related source code recommendation to infer the legality of targeted warnings via GPT-C [91]. Similar to the usage of CodeBERTa in the work of Kharkar et al., Yedida et al. [112] train CodeBERT [13] on the labeled warnings and use the well-trained CodeBERT for AWI.

As shown in Fig. 6a, 12 studies attempt multiple model categories for AWI. Specifically, 10 studies [31, 46, 47, 62, 76, 78, 90, 94, 109, 113] separately attempt TML and DL models for AWI. One study [41] separately uses TML model and PTMs for AWI. One study [112] uses TDP for AWI respectively.

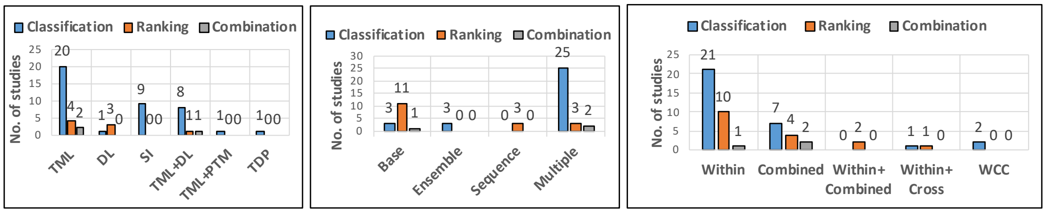
4.5.2 Learning category in AWI. Almost all primary studies, except for the work of Tu et al. [94], perform AWI in a supervised learning manner. That is, these studies train an AWI model in the labeled warnings and use this model for unlabeled warnings. By contrast, Tu et al. adopt semi-supervised learning for AWI. Specifically, they rely on an unsupervised learning technique called CLAMI [69] to assign pseudo-labels for warnings via clustering, select warning features via metric violation score, and determine warning instances for the AWI model training. However, CLAMI has many hyperparameters, which greatly affect AWI performance. To address this problem, they search for the optimal hyperparameters on a few labeled warnings. Under the optimal hyperparameters, they finally use the determined warning instances with the associated pseudo warning labels and the selected warning features to train a classifier for AWI.

4.5.3 AWI model structure. By analyzing the primary studies, the AWI model involves the base, ensemble, sequence, and multiple models-based structures.

Base structure. The base structure uses a single ML model for AWI. Fig. 6b describes that 15 studies only rely on the base structure for AWI, where 3 [32, 51, 85], 11 [9, 33, 36, 44, 49, 50, 56, 61, 70, 80, 120], and 1 [115] studies fall into classification, ranking, and combination approaches respectively. The base structure is easy to implement and deals with massive warnings with high-dimensional features. However, the base structure could face the overfitting problem.

Table 7. Distribution of the model category. The full names of model abbreviations are shown below: Random Forest (RF), Decision Tree (DT), Naive Bayes (NB), Support Vector Machine (SVM), Logistical Regression (LR), K-Nearest Neighbor (KNN), Bayesian Network (BN), KStar (K*), Ensemble (the customized model based on the idea of ensemble learning), Random Committee (RC), Random Tree (RT), Others (i.e., PART, Ridor, Conjunctive Rule, ADTree, REPTree, LMT, LWL, IBK, and Decision Table), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), MultiLayer Perceptron (MLP), Deep Neural Network (DNN), Bi-directional Gating Recurrent Unit (BiGRU), Gated Graph Neural Network (GGNN), Bi-directional Long Short-Term Memory (BiLSTM), Artificial Neural Network (ANN), and Markov Logic Network (MLN).

Model category	**Model**	**Approaches**			**Studies**
		Classification	**Ranking**	**Combination**	
TML model	RF(22)	21	0	1	[3, 5, 14, 15, 18, 31, 47, 62, 78, 90, 94, 102, 105, 107, 109, 110, 113, 114, 117, 118, 122, 123]
	DT(21)	15	4	2	[5, 18, 26, 27, 31, 47, 62, 76, 79, 82, 83, 90, 93, 102, 105, 107, 109, 110, 112, 113, 117]
	SVM(18)	15	1	2	[3, 18, 39, 47, 62, 63, 79, 90, 93, 94, 102, 107, 109, 110, 112, 113, 115]
	NB(17)	14	1	2	[18, 26, 27, 31, 46, 47, 62, 63, 78, 79, 90, 93, 102, 107, 113, 114, 117]
	LR(13)	12	1	0	[5, 14, 15, 18, 27, 41, 62, 79, 85, 102, 107, 112, 114]
	KNN(9)	7	2	0	[3, 18, 39, 56, 62, 63, 78, 79, 107]
	Boosting(9)	7	2	0	[14, 15, 18, 63, 82, 83, 102, 114, 123]
	BN(6)	5	0	1	[26, 27, 47, 62, 93, 113]
	K*(4)	4	0	0	[27, 47, 93, 113]
	OneR(3)	3	0	0	[47, 93, 113]
	ZeroR(3)	3	0	0	[47, 93, 113]
	Ensemble(3)	3	0	0	[118, 122, 123]
	RC(2)	2	0	0	[117, 118]
	LightGBM(2)	2	0	0	[14, 123]
	NDTree(1)	1	0	0	[93]
	Ripper (1)	1	0	0	[3]
	DTNB(1)	1	0	0	[118]
	RT(1)	1	0	0	[62]
	Others(1)	1	0	0	[27]
DL model	CNN(6)	4	2	0	[51, 78, 92, 98, 109, 112]
	LSTM(4)	4	0	0	[46, 47, 78, 113]
	MLP(4)	3	0	1	[47, 62, 90, 113]
	DNN(2)	2	0	0	[31, 109]
	BiGRU(2)	0	1	0	[98]
	GGNN(2)	2	0	0	[47, 113]
	BiLSTM(2)	0	2	0	[70, 92]
	ANN(3)	2	1	0	[76, 94, 112]
Statistical inference model	Bayesian inference (7)	0	7	0	[9, 33, 36, 44, 49, 61, 80]
	Z-test(1)	0	1	0	[50]
	MLN(1)	0	1	0	[120]
PTM	CodeBERTa(1)	1	0	0	[41]
	CodeBERT(1)	1	0	0	[112]
	GPT-C(1)	1	0	0	[41]



(a) Model categories in AWI. (b) AWI model structures. (c) AWI construction scenarios.

Fig. 6. Distribution of model categories in AWI, AWI model structures, and AWI construction scenarios in three categories of ML-based AWI approaches. SI is the statistical inference. TDP is TML+DL+PTM. WCC is to separately adopt within, combined, cross projects for construction scenarios for AWI in the same study.

Ensemble structure. The ensemble structure relies on bagging, boosting, or stacking to combine multiple ML models for AWI. Different from the off-the-shelf ensemble learning models (e.g., RF), the ensemble structure borrows the core idea of ensemble learning and independently designs a new classifier/sorter by combining multiple ML models. Fig. 6b shows that three studies [118, 122, 123] related to the classification approach adopt the ensemble structure for AWI. For example, D2A [123] applies a soft-voting strategy to combine the results of four ML models for AWI. Compared to the base structure, the ensemble structure can reduce the overfitting problem to some extent, obtain higher AWI performance, and be more stable and reliable. However, the ensemble structure is slower than the base structure when encountering massive warnings with high-dimensional features because each ML model in the ensemble structure needs to conduct an AWI model.

Sequence structure. The studies using the sequence structure divide a complex problem into sub-problems and sequentially solve each problem by using a proper ML model. Fig. 6b shows that three studies [82, 83, 98] related to the ranking approach use the sequence structure for AWI. For example, given the control flow-based abstract syntax tree and program slice of warnings as input, Vu et al. [98] use CNN to encode these features, then use BiGRU to further learn the distribution of results encoded by CNN, and finally use dense layers to prioritize warnings. The sequence structure could identify more complex warnings while being slower than base and ensemble structures. Also, it requires further exploration about reasonably partitioning the AWI problem into sub-problems and deliberately selecting proper ML models for sequentially solving each sub-problem.

Multiple models-based structure. The multiple models-based structure uses different ML models for AWI each time, thereby helping search for the optimal AWI performance. Fig. 6b shows that the multiple models-based structure involves the most studies, where 25 [3, 5, 14, 15, 18, 27, 31, 39, 41, 46, 47, 62, 63, 78, 93, 94, 102, 105, 107, 109, 110, 112–114, 117], 3 [76, 79, 92], and 2 [26, 90] studies are related to classification, ranking, and combination approaches respectively. The multiple models-based structure can help investigate the performance differences of multiple ML models for the AWI task. However, since the same task requires repeating multiple times, the multiple models-based structure is generally more time-consuming than the above three structures.

4.5.4 AWI construction scenario. By analyzing the primary studies, the AWI construction scenario can be summarized into three categories, including within, combined, and cross projects.

Within project. The within project is that the warning dataset used for the model construction (i.e., the training set), and model evaluation (i.e., the test set) is from the same project. It indicates that the training and test sets have a similar distribution, thereby yielding superior AWI performance. However, the number of the training set is frequently insufficient. As shown in Fig. 6b, 63% (32/51) of primary studies only use the within project as the AWI construction scenario. Of these, 21 [3, 5, 14, 15, 18, 27, 39, 46, 63, 78, 85, 94, 105, 107, 109, 110, 112, 117, 118, 122, 123], 10 [9, 33, 36, 50, 56, 61, 79, 82, 83, 120], and 1 [26] studies fall into the classification, ranking, and combination approaches respectively.

Combined project. The combined project merges warnings in multiple projects as a whole and partitions the merged warning dataset into the training and test sets. Fig. 6b shows that 13 studies separately adopt the combined project as the AWI construction scenario, where seven [31, 32, 41, 51, 62, 93, 114], four [44, 76, 80, 92], two [90, 115] studies are related to classification, ranking, and combination approaches respectively. The difference between within and combined project-based construction scenarios is that the warning dataset in the former is from the same project, and the warning dataset in the latter is merged from multiple projects. The combined project can alleviate the weakness of the within project to some extent, while disrupting the original distribution of warning dataset in the within project.

Cross project. The cross project is that the warning dataset used for the model construction (i.e., the training set) and model evaluation (i.e., the test set) is from different projects. It indicates that the training and test sets are heterogeneous. Only two studies [96, 102] separately adopt the cross project as the AWI construction scenario. The cross project can greatly mitigate the problem of warning dataset sparsity in the within project. However, due to the heterogeneous distribution in the training and test sets, it is more difficult to obtain the high AWI performance in the cross project than in the within project.

As shown in Fig. 6b, six studies attempt multiple categories of AWI construction scenarios. Specifically, two [70, 98], two [49, 102], and two [47, 113] studies separately adopt within and combined projects, within and cross projects, and WCC projects for AWI.

Summary RQ4: The AWI model construction stage in the ML-based AWI approaches involves model category, learning category, AWI model structure, and AWI construction scenario. Specifically, ML models used for AWI are almost based on supervised learning, where the TML model occupies the majority. The multiple models-based structure is the most commonly used for AWI. The within project is the most prevalent AWI construction scenario.

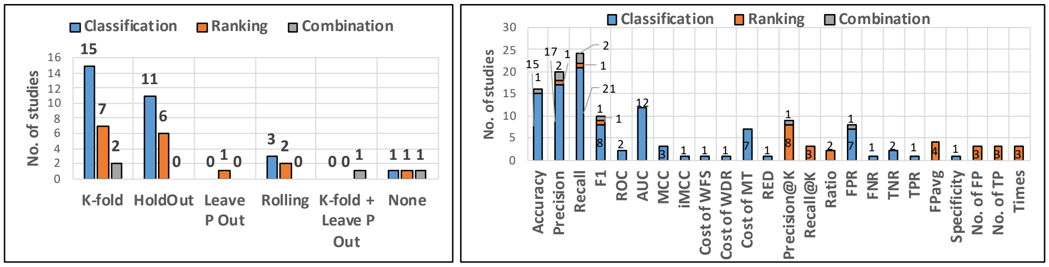
4.6 RQ5: How are ML-based AWI approaches evaluated?

The ML-based AWI model evaluation is critical for the practical applicability of ML techniques in AWI. Based on the typical ML-based AWI workflow in Fig. 2, the AWI model evaluation mainly involves two parts, including validation strategy and performance measure. To present details in the AWI model evaluation stage, we analyze the techniques used in the two parts, along with associated strengths, weaknesses, and distribution across three categories of ML-based AWI approaches.

4.6.1 *Validation strategy in AWI.*

By analyzing primary studies, the validation strategy to evaluate ML-based AWI performance includes K-fold, HoldOut, Leave P Out, and rolling cross validation.

***K*-fold cross validation**. K-fold cross validation divides the warning dataset into K equally-sized folds. The ML model is trained and tested K times, where a fold is used for the test set and the remaining K-*1* folds are used for the training set each time. In general, K is set to 10 [18, 27, 79, 105] or 5 [41, 47, 70, 92, 98, 113]. Fig. 7a shows that 24 studies adopt K-fold cross validation, where 15 [3, 5, 18, 27, 41, 47, 51, 62, 63, 105, 107, 113, 117, 118, 122], 7 [70, 76, 79, 82, 83, 92, 98], 2 [26, 90] studies involve classification, ranking, and combination approaches respectively. In K-fold cross validation, the entire warning dataset is used as the training and test sets, which can mitigate the estimation bias [48]. However, for the imbalanced warning dataset, K-fold cross validation may not ensure that each fold maintains the same class proportions as the whole warning dataset, thereby biasing the ML-based AWI performance [86]. Also, due to the iterative nature, K-fold cross validation has high computational time.



(a) Validation strategies.

(b) Performance measures.

Fig. 7. Distribution of validation strategies and performance measures under the three categories of ML-based AWI approaches. MCC is Matthews Correlation Coefficient. iMCC [18] is the MCC difference between two approaches. Cost for WFS, WDR, and MT denote the time of warning feature selection, warning dataset rebalancing, and model training, respectively. RED [107] is the redundancy rate of warning features. Ratio [49, 83] is the performance ratio of the proposed ranking approach against the random ranking. FP_{avg} is the average cumulative number of unactionable warnings before actionable warnings are found [49, 79, 82, 83]. Times are the iterations for all discovered bugs in the ML-based AWI ranking approach.

HoldOut cross validation. HoldOut cross validation randomly divides the warning dataset into the training and sets. The common split ratios in HoldOut cross validation are 70/30 [15, 85] or 80/20 [46, 85, 112]. Fig. 7a shows that 17 studies adopt HoldOut cross validation, where 11 [14, 15, 31, 46, 85, 93, 94, 109, 112, 114, 123] and 6 [9, 36, 49, 61, 80, 120]) studies are related to classification and ranking approaches respectively. In comparison to *K*-fold cross validation, HoldOut cross validation can finish the AWI model evaluation faster. However, HoldOut cross validation highly relies on the class distribution and split ratio, which could cause overfitting [86].

Leave *P* Out cross validation. In Leave *P* Out cross validation, *P* warnings are left as the test set, and the remaining warnings are used as the training set. This process is repeated for all possible combinations of leaving out *P* samples from the warning dataset. Only one study [44] related to the warning classification adopts Leave *P* Out cross validation. Similar to *K*-fold cross validation, the entire warning dataset in Leave *P* Out cross validation is used for the training and test sets, which can reduce the estimation bias [48]. Compared to HoldOut cross validation, Leave *P* Out cross validation is more time-consuming when used for the AWI model evaluation.

Rolling cross validation. Rolling, aka time series-based, cross validation divides the warning dataset into the training and test sets in chronological order. For example, Wang et al. [39] build an AWI classifier in the warnings reported from a prior revision and use the warnings reported from a later revision to evaluate the performance of this classifier. Fig. 7a shows that five studies adopt the rolling cross validation, where three [39, 102, 110] and two [33, 56] studies fall into classification and ranking approaches respectively. Rolling cross validation can reflect the real-world validation scenario, while being incompatible with the above three strategies due to time-sensitivity.

As shown in Fig. 7a, one study [32] separately attempts K-fold and HoldOut cross validation for AWI model evaluation. In addition, three studies [50, 78, 115] do not clearly mention the validation strategies for AWI model evaluation.

4.6.2 Performance measure in AWI. As shown in Fig. 7b, 47% (24/51) of primary studies use Recall, followed by Precision (39%) and Accuracy (31%). Precision@K and FP_{avg} are the most frequently used by the ranking approach, which involve eight [36, 50, 56, 70, 76, 80, 92, 98] and four [49, 79, 82, 83] studies respectively. Nine studies use Cost to evaluate the efficiency of sub-parts in the classification approach, including the warning feature selection [107], warning dataset rebalancing [18], and model training [46, 47, 85, 102, 109, 110, 113]. In particular, one study [107] designs RED to evaluate the performance difference among warning feature selection techniques. One study [18] uses iMCC, derived from MCC, to evaluate the MCC difference between two ML-based AWI approaches. Two studies [49, 83] adopt Ratio to evaluate the performance difference between the proposed ranking approach and random ranking.

Since different studies conduct experimental evaluations on different warning datasets, we focus on analyzing the performance results in 10 studies with the same warning dataset. It is observed that on the same noisy and duplicate warning samples with data-leaked warning features, the AUC of eight studies [18, 79, 94, 102, 105, 107, 109, 110] almost falls into 70%~90%. However, on the same clean warning features and samples obtained via the manual filtering in the work of Kang et al. [39], AUC generally drops to 50%~70% [39, 107, 112]. This means that there is the substantial room for AWI performance improvement.

Summary RQ5: The AWI model evaluation stage in the ML-based AWI approaches contains the validation strategy and performance measure. Specifically, K-fold cross validation occupies the majority, followed by HoldOut and rolling cross validation. Recall, Precision, and Accuracy are the most commonly used performance measures in primary studies.

5 Future Research Directions

Based on the above analysis results in primary studies, we highlight research directions for the future ML-based AWI field from the perspectives of data improvement and model exploration in the typical ML-based AWI workflow.

5.1 Research Directions in Data Improvement

(1) Collecting warning datasets with various SCAs and project development languages. As shown in Section 4.3, most primary studies collect warning datasets from FindBugs and CppCheck as well as Java and C/C++ projects, respectively. However, few studies focus on SCAs (e.g., the symbolic execution-based SCA called Klocwork[4]) and project development languages (e.g., Python) for the warning dataset acquisition. Also, the analysis results in Section 4.3 indicate that due to SCAs with various techniques and project development languages, the acquired warning dataset is vastly diverse. Such diversity can help evaluate the generalizability of the ML-based AWI approach [57, 116]. In addition, it is observed that the warning datasets in only 17 primary studies are traceable. Specifically, the most commonly used and available real-world warning dataset, collected from FindBugs and Java projects, has been adopted by 10 studies [18, 39, 79, 94, 102, 105, 107, 109, 110, 112]. However, such a warning dataset contains mislabeled and duplicate samples [39]. The warning datasets used in two studies [47, 113] can be available. However, the warning datasets contain many synthetic warnings and only a small part of real-world warnings. In particular, there are only about 400 real-world warnings used for AWI in the study [113]. The remaining studies [27, 61, 76, 80, 120] either only disclose the real-world projects rather than the warning dataset corresponding with these projects, or collect warning datasets from the synthetic project source. Thus, it is essential to collect the diverse and real-world warning dataset and encourage disclosure of the collected warning dataset to stimulate the ML-based AWI process.

(2) Improving the labeling strategy to construct more reliable warning datasets. It is summarized in Section 4.3 that the automatic and manual strategies are two main ways to label warnings. The automatic labeling strategy, especially the closed warning-based heuristic [39], is an important tactic to automatically construct a large-scale and real-world dataset. However, such a heuristic is extremely sensitive to the warning-irrelevant source code changes, thereby causing mislabeled warnings [102]. Although some studies [53, 57, 116] are proposed to improve such a heuristic, there is still much improvement room for the warning labeling accuracy due to limitations of the warning-irrelevant source code discernment algorithm. It indicates that the current

[4] https://www.perforce.com/products/klocwork

state-of-the-art heuristic is not still robust enough to label warnings. By contrast, solely performing the warning labeling via the manual strategy is not easy to scale and may be subject to the bias of an annotator [39]. Despite combining manual and automatic strategies for warning labeling in the primary studies, the current hybrid strategy mainly focuses on serialization (i.e., the manual and automatic strategies are used in sequential order to label warnings) rather than collaboration (i.e., the manual and automatic strategies are used interactively to label warnings). Such a serialization way results in the inability to fully utilize the role of manual and automatic strategies in the warning labeling process. Thus, we believe that it could be a promising labeling strategy via the human-machine collaboration to construct a reliable warning dataset. That is, the heuristic can gather different information (e.g., the source code revision message and the developer activity) to enrich warnings, thereby helping simplify the manual labeling process for annotators. In turn, the manual annotation could provide domain knowledge for complex warnings, thereby helping the heuristic obtain more precision warning labels.

(3) Characterizing warnings with more fine-grained features for AWI. First, Section 4.4 shows five categories of warning features in the primary studies. A few studies [41, 47, 113] only mention that the sequential warning features can achieve superior performance in comparison to the characteristic-based and statistical warning features. However, it is not unclear which of the five warning feature categories in Section 4.4 shows the most powerful AWI ability. Further, although some studies (e.g., [102]) attempt different categories of warning features (e.g., characteristic-based, statistical, and content-based) for AWI, these studies are unaware of what combinations of different warning feature categories could maximize AWI performance. Thus, it is necessary to conduct a comprehensive empirical evaluation on different categories of warning features and their combinations, thereby seeking out more precise and thorough warning features for AWI. Second, in comparison to the other four categories, the structural category is the most expressive in the root cause of actionable and unactionable warnings due to being able to grasp the intrinsic information of reported warnings (i.e., syntax and semantics) [2]. In nine studies related to the structural category, warning features are mainly extracted by using the program slicing [104] to obtain the program dependency graph of the warning or using the def-use analysis [73] to construct the datalog derivation graph of the warning. However, due to the limitations of program slicing and def-use analysis techniques [9, 73, 104], there is still warning-irrelevant or imprecision information in the program dependency graph and datalog derivation graph. Thus, it is essential to use more precise static analysis (e.g., SMT [37]) and more explicit dynamic execution tactics (e.g., fuzzing [38]) for acquiring sufficiently structural information but eliminating irrelevant information, thereby extracting more rigorous and complete warning features for AWI. Third, the SCA report is one of the important warning feature sources. The warning message (i.e., a warning characteristic in the SCA report) summarizes the basic warning information, which can provide auxiliary information for AWI [81]. However, few studies consider the warning message for AWI. In the future, it could be a useful way to further enrich warnings for AWI by incorporating the warning message into warning features.

(4) Employing feature selection techniques to enhance the AWI performance. Through the fractal-based method [52], Yang et al. [109] experimentally prove that the original warning features in the work of Wang et al. [102] are inherently low-dimension. Similarly, Ge et al. [107] observe that the original warning features [112] contain irrelevant and redundant features via PCA. Also, the evaluation in five studies [3, 27, 102, 117, 123] explicitly reveals that the warning features processed by feature selection techniques are more discriminative for AWI. In particular, instead of using the off-the-shelf feature selection techniques [3, 27, 102, 117, 123], UNEASE[107] designs an unsupervised warning feature selection method for AWI, which can obtain the top-ranked AUC while maintaining low feature selection cost and feature redundancy rate. The above findings demonstrate that there could be irrelevance and redundancy in the original warning features, and the AWI performance can be further amplified after applying feature selection techniques to the original warning features. However, most primary studies directly adopt the original warning features for AWI, while only 11 studies adopt feature selection techniques for AWI. Thus, researchers and practitioners should pay more attention to the role of feature selection techniques in AWI. Further, it could be useful to employ feature selection techniques for enhancing AWI performance.

(5) Selecting elaborately class rebalancing techniques to improve the AWI performance. The warning dataset in the primary studies presents a prevailing phenomenon, i.e., class imbalance. Such a phenomenon limits the ML-based AWI performance [18, 123]. However, most primary studies (i.e., 43) ignore the class imbalance when performing ML-based AWI. Only nearly 16% (8/51) of primary studies attempt the class rebalancing technique to mitigate the class imbalance, thereby performing ML-based AWI. In particular, Ge et al. [18] investigate whether the off-the-shelf class rebalancing techniques can consistently improve the ML-based AWI performance. The experimental results describe that most class rebalancing techniques can significantly improve ML-based AWI performance. Surprisingly, a small part of class rebalancing techniques (e.g., KMeans-SMOTE) does not work for AWI in the imbalanced warning datasets. As such, researchers and practitioners should be concerned about the impact of class imbalance on AWI performance and further select elaborately the class rebalancing technique to improve AWI performance.

5.2 Research Directions in Model Exploration

(1) Attempting semi-supervised or unsupervised learning for AWI. The ML paradigm mainly contains supervised, semi-supervised, and unsupervised learning. As shown in Section 4.5, almost all ML-based AWI approaches in the primary studies follow supervised learning for AWI, which often requires massive labeled warnings to achieve high AWI performance. However, as shown in the warning dataset preparation of Section 4.3, massive reliable and labeled warnings are often hard to gather quickly in practice [39, 53, 57, 116]. By contrast, semi-supervised learning can use a small portion of labeled samples and lots of unlabeled samples to train a predictive model, and unsupervised learning can identify patterns in unlabeled samples. Particularly, Tu et al. [94] is the first to adopt semi-supervised learning for AWI and achieve superior AWI performance. Thus, semi-supervised or unsupervised learning is strongly recommended for AWI, so as to help reduce the number of labeled warnings as many as possible while maintaining high AWI performance.

(2) Exploring AWI via large language models. Large Language Models (LLMs) are PTMs with larger-scale corpora and more training parameters. Similar to PTMs, LLMs can directly capture the rich sequential, syntactic, and semantic information from a given source code snippet. This is because the well-designed attention mechanism in the transformer structure of LLMs has the ability to localize areas of interest, thereby helping explain the factors contributing to specific tasks (e.g., vulnerability detection) and effectively boosting their performance [60, 99]. Besides, pre-training (i.e., training on large-scale and unlabeled corpora) and fine-tuning (i.e., training on a few labeled samples related to the downstream task) are proved to play critical roles in LLMs [19]. Currently, LLMs have already exhibited tremendous potential in various software engineering tasks [58]. However, only two studies [41, 112] attempt PTMs for AWI. Despite revealing higher AWI performance in PTMs compared to TML and DL models, Kharkar et al. [41] only conduct a preliminary study for AWI on two obsolete PTMs (i.e., CodeBERTa and GPT-C). It indicates that the ability of current LLMs (e.g., ChatGPT4[5]) has not been fully investigated in the ML-based AWI community. As many state-of-the-art LLMs are proposed and released, an immediate direction is to conduct systematic experiments to understand the merits and shortcomings of LLMs in AWI, thereby exploring how to use LLMs for enhancing AWI. In addition, it could be an effective way to enlarge the benefits of pre-training (e.g., developing domain-specific LLMs by pre-training the AWI-related task) and fine-tuning (e.g., enhancing AWI performance by fine-tuning LLMs on massive warning datasets) when applying LLMs to AWI.

(3) Designing the customized AWI approaches for different categories of warnings. Different categories of warnings have different characteristics. For example, warnings with cross-site scripting (XSS) generally contain

[5] https://chat.openai.com/

URI schema, host name, or port number. Once an XSS vulnerability in a project is attacked, the privacy information of this project is leaked. By contrast, warnings with bad practices are related to the source code writing standards, which do not necessarily cause software defects. This indicates that an ML-based AWI approach used to identify one category of warnings could not perfectly fit the other category of warnings. However, nearly 65% (33/51) of primary studies ignore the impact of warning categories on AWI, i.e., adopt a designed ML-based AWI approach to sweepingly identify all categories of warnings reported by a SCA. Although the remaining 18 primary studies [3, 9, 14, 15, 32, 33, 41, 44, 49, 51, 61, 62, 70, 76, 80, 92, 98, 120] design the corresponding ML-based AWI approaches to identify one or several specific warning categories (e.g., datatrace [80]), these studies ignore the exploration about the performance difference of the designed ML-based AWI approach on different categories of warnings. In future work, it is necessary to make the cons and pros of different ML-based AWI approaches on different categories of warnings, thereby helping select a proper ML-based AWI approach for a given warning category. Further, it is strongly recommended to design the customized ML-based AWI approach for different categories of warnings, thereby contrapuntally enhancing AWI performance.

(4) Conducting the practical AWI model evaluation via rolling cross validation. As shown in Section 4.6, the common strategy used for the ML-based AWI model evaluation is K-fold and HoldOut cross validation, which occupy 80% of primary studies. The two validation strategies, especially for K-fold cross validation, often make the sample distribution between the training and test sets consistent [20]. Conversely, in the real-world scenario, warnings in the training set should be historically prior to that of the test set. It indicates that warnings in the test set may be similar to warnings in the training set, contain warnings that are unknown to the training set, or even be entirely different from warnings in the training set. Thus, the two validation strategies ignore the warning timelines in the real-world scenario, which could result in exaggerating the performance of ML-based AWI approaches [20, 75]. The rolling cross validation, dividing the warning dataset into the training and test sets in chronological order, can exactly satisfy the ML-based AWI model evaluation requirement in the real-world scenario. However, only five primary studies [33, 39, 56, 102, 110] consider the rolling cross validation as the ML-based AWI validation strategy. It indicates that the performance of ML-based AWI approaches in most primary studies could be biased in the real-world scenario. Thus, we recommend adopting the rolling cross validation to conduct the practical AWI model evaluation, thereby helping reveal the ML-based AWI performance in the real-world scenario.

6 Threats to Validity

External. The threat to external validity concerns whether the findings of our survey are generalizable beyond studies in the entire ML-based AWI population. Since all primary studies are collected from the entire ML-based AWI population, this threat is not applicable.

Internal. The threat to internal validity concerns the problems of data extraction consistency and correctness. To alleviate the above problems, we conduct a pilot study and double manual verification in the process of data extraction (see in Section 2).

Construct. The threat to construct validity concerns whether our collected primary studies are related to ML-based AWI or miss relevant ML-based AWI studies. To ensure the relevance and completeness of the collected primary studies, we first inherit and extend the search keywords from the latest AWI studies [67]. Besides, we design the selection criteria based on a pilot search and perform the selection of primary studies based on the 2-pass review. Moreover, we conduct the snowballing to help identify relevant studies that could be missed by the keywords-based search. Thus, we believe that this threat of construct validity can be mitigated in our survey.

7 Conclusion

In this paper, we conduct a comprehensive survey related to the ML-based AWI approach. We first perform a meticulous survey methodology to collect ML-based AWI studies in five digital libraries from 2000 to 2023, thereby obtaining 51 primary studies. After that, we describe a typical ML-based AWI workflow, including the warning dataset preparation, preprocessing, AWI model construction, and evaluation. By following such a workflow, we rely on the warning output format to classify ML-based AWI approaches into three categories, involving classification, ranking, and combination approaches. Besides, we analyze the techniques used in each stage of such a typical workflow by discussing their strengths and weaknesses as well as presenting their distribution under the three ML-based AWI approach categories. Finally, we rely on the analysis results to highlight practical research directions for the ML-based AWI community.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments. The work is partly supported by the National Natural Science Foundation of China (61932012, 62372228, 62141215) and the Program of China Scholarship Council (Grant No. 202306190140).

References

[1] Jens Grabowski Alexander Trautsch, Steffen Herbold. 2020. A longitudinal study of static analysis warning evolution and the effects of pmd on software quality in Apache open source projects. *Empirical Software Engineering (EMSE)* 25 (2020), 5137–5192.

[2] Artemis Alexiadou, Hagit Borer, and Florian Schäfer. 2014. *The syntax of roots and the roots of syntax*. Vol. 51. Oxford University Press.

[3] Enas A. Alikhashashneh, Rajeev R. Raje, and James H. Hill. 2018. Using machine learning techniques to classify and predict static sode snalysis tool warnings. In *Proceedings of the 15th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*. IEEE, Aqaba, 1–8.

[4] Esben Sparre Andreasen, Anders Møller, and Benjamin Barslev Nielsen. 2017. Systematic approaches for increasing soundness and precision of static analyzers. In *Proceedings of the 6th ACM SIGPLAN International Workshop on State of the Art in Program Analysis*. ACM, Barcelona, 31–36.

[5] Maxwell Berman, Stephen Adams, Tim Sherburne, Cody Fleming, and Peter Beling. 2019. Active learning to improve static analysis. In *Proceedings of the 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, Florida, 1322–1327.

[6] Tim Buckers, Clinton Cao, Michiel Doesburg, Boning Gong, Sunwei Wang, Moritz Beller, and Andy Zaidman. 2017. UAV: warnings from multiple automated static analysis tools at a glance. In *Proceedings of the 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Klagenfurt, 472–476.

[7] David Budgen and Pearl Brereton. 2006. Performing systematic literature reviews in software engineering. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Shanghai, 1051–1052.

[8] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. 2017. Interpretability of deep learning models: a survey of results. In *IEEE Smartworld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (Smartworld/SCALCOM/UIC/ATC/CBDcom/IOP/SCI)*. IEEE, San Francisco, 1–6.

[9] Tianyi Chen, Kihong Heo, and Mukund Raghothaman. 2021. Boosting static analysis accuracy with instrumented test executions. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESE/FSE)*. ACM, Singapore, 1154–1165.

[10] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. ACM, Los Angeles California, 238–252.

[11] Thomas Durieux, João F. Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the 42rd International Conference on Software Engineering (ICSE)*. IEEE/ACM, Seoul, 530–541.

[12] Dawson Engler, Benjamin Chelf, Andy Chou, and Seth Hallem. 2000. Checking system rules using system-specific, programmer-written compiler extensions. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*. ACM, San Diego, 1–16.

[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *Findings of the Association for Computational Linguistics*

(ACL) 1, 1 (2020), 1536–1547.

[14] Lori Flynn, William Snavely, and Zachary Kurtz. 2021. Test suites as a source of training data for static analysis alert classifiers. In *Proceedings of the 2nd IEEE/ACM International Conference on Automation of Software Test (AST)*. ACM, Seoul, 100–108.

[15] Lori Flynn, William Snavely, David Svoboda, Nathan VanHoudnos, Richard Qin, Jennifer Burns, David Zubrow, Robert Stoddard, and Guillermo Marce-Santurio. 2018. Prioritizing alerts from multiple static analysis tools, using classification models. In *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies (SQUADE)*. IEEE/ACM, Gothenburg, 13–20.

[16] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. 2012. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 4 (2012), 463–484.

[17] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–37.

[18] Xiuting Ge, Chunrong Fang, Tongtong Bai, Jia Liu, and Zhihong Zhao. 2023. An empirical study of class rebalancing methods for actionable warning identification. *IEEE Transactions on Reliability (TR) (Early Access)* 72, 4 (2023), 1–15.

[19] Xiuting Ge, Chunrong Fang, Quanjun Zhang, Daoyuan Wu, Bowen Yu, Qirui Zheng, An Guo, Shangwei Lin, Zhihong Zhao, Yang Liu, et al. 2024. Pre-trained Model-based Actionable Warning Identification: A Feasibility Study. *arXiv preprint arXiv:2403.02716* (2024).

[20] Xiuting Ge, Yifan Huang, Zhanwei Hui, Xiaojuan Wang, and Xu Cao. 2021. Impact of datasets on machine learning based methods in Android malware detection: an empirical study. In *Proceedings of the 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Hainan, 81–92.

[21] Xiuting Ge, Shengcheng Yu, Chunrong Fang, Qi Zhu, and Zhihong Zhao. 2023. Leveraging android automated testing to assist crowdsourced testing. *IEEE Transactions on Software Engineering (TSE)* 49, 4 (2023), 2318–2336.

[22] Asem Ghaleb and Karthik Pattabiraman. 2020. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, Vitual Event USA, 415–427.

[23] Trisha Greenhalgh and Richard Peacock. 2005. Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources. *Bmj* 331, 7524 (2005), 1064–1065.

[24] Zhaoqiang Guo, Tingting Tan, Shiran Liu, Xutong Liu, Wei Lai, Yibiao Yang, Yanhui Li, Lin Chen, Wei Dong, and Yuming Zhou. 2023. Mitigating False Positive Static Analysis Warnings: Progress, Challenges, and Opportunities. *IEEE Transactions on Software Engineering (TSE)* 49, 12 (2023), 5154–5188.

[25] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.

[26] Quinn Hanam, Lin Tan, Reid Holmes, and Patrick Lam. 2014. Finding patterns in static analysis alerts: improving actionable alert ranking. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, Hyderabad, 152–161.

[27] Sarah Heckman and Laurie Williams. 2009. A model building process for identifying actionable static analysis alerts. In *Proceedings of the 2rd International Conference on Software Testing Verification and Validation (ICST)*. IEEE, Dublin, 161–170.

[28] Sarah Heckman and Laurie Williams. 2011. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology (IST)* 53, 4 (2011), 363–387.

[29] Sarah Smith Heckman. 2007. Adaptively ranking alerts generated from automated static analysis. *XRDS: Crossroads, The ACM Magazine for Students* 14, 1 (2007), 1–11.

[30] Sarah Smith Heckman and Laurie Ann Williams. 2008. A measurement framework of alert characteristics for false positive mitigation models. *North Carolina State University, Depth of Computer Science* 1 (2008), 1–6.

[31] Péter Hegedűs and Rudolf Ferenc. 2022. Static code analysis alarms filtering reloaded: a new real-world dataset and its ml-based utilization. *IEEE Access* 10 (2022), 55090–55101.

[32] Kihong Heo, Hakjoo Oh, and Kwangkeun Yi. 2017. Machine-learning-guided selectively unsound static analysis. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Buenos Aires, 519–529.

[33] Kihong Heo, Mukund Raghothaman, Xujie Si, and Mayur Naik. 2019. Continuously reasoning about programs using differential bayesian inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, Copenhagen, 561–575.

[34] Apirak Hoonlor, Boleslaw K Szymanski, and Mohammed J Zaki. 2013. Trends in computer science research. *Communiccations of the ACM* 56, 10 (2013), 74–83.

[35] James W. Hunt and Thomas G. Szymanski. 1977. A fast algorithm for computing longest common subsequences. *Commun. ACM* 20, 5 (1977), 350–353.

[36] Yungbum Jung, Jaehwang Kim, Jaeho Shin, and Kwangkeun Yi. 2005. Taming false alarms from a domain-unaware c analyzer by a bayesian statistical post analysis. In *Proceedings of the 12th International Conference on Static Analysis (SAS)*. ACM, Berlin, 203–217.

[37] Maximilian Junker, Ralf Huuck, Ansgar Fehnker, and Alexander Knapp. 2012. SMT-based false positive elimination in static program analysis. In *Proceeding of the 14th International Conference on Formal Engineering Methods (ICFEM)*. IEEE, Kyoto, 316–331.

[38] Ashwin Kallingal Joshy, Xueyuan Chen, Benjamin Steenhoek, and Wei Le. 2021. Validating Static Warnings via Testing Code Fragments. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, Denmark, 540–552.

[39] Hong Jin Kang, Khai Loong Aw, and David Lo. 2022. Detecting false alarms from automatic static analysis tools: how far are we?. In *Proceedings of the 44th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE/ACM, Pittsburgh, 698–709.

[40] Staffs Keele et al. 2007. Guidelines for performing systematic literature reviews in software engineering.

[41] Anant Kharkar, Roshanak Zilouchian Moghaddam, Matthew Jin, Xiaoyu Liu, Xin Shi, Colin Clement, and Neel Sundaresan. 2022. Learning to reduce false positives in analytic bug Detectors. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Pittsburgh, 1307–1316.

[42] Jack Kiefer and Jacob Wolfowitz. 1959. Optimum designs in regression problems. *The annals of mathematical statistics* 30, 2 (1959), 271–294.

[43] Gary A. Kildall. 1973. A unified approach to global program optimization. In *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, Boston, 194–206.

[44] Hyunsu Kim, Mukund Raghothaman, and Kihong Heo. 2022. Learning probabilistic models for static analysis alarms. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Pittsburgh, 1282–1293.

[45] Sunghun Kim and Michael D. Ernst. 2007. Which warnings should i fix first?. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*. ACM, Dubrovnik, 45–54.

[46] Ugur Koc, Parsa Saadatpanah, Jeffrey S. Foster, and Adam A. Porter. 2017. Learning a classifier for false positive rrror reports emitted by static code analysis tools. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*. ACM, Barcelona, 35–42.

[47] Ugur Koc, Shiyi Wei, Jeffrey S. Foster, Marine Carpuat, and Adam A. Porter. 2019. An empirical assessment of machine learning approaches for triaging reports of a java static analysis tool. In *Proceedings of the 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, Xi'an, 288—99.

[48] Ron Kohavi et al. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 2rd of International Joint Conference on Artificial Intelligence*, Vol. 14. IEEE, Montreal, 1137–1145.

[49] Ted Kremenek, Ken Ashcraft, Junfeng Yang, and Dawson Engler. 2004. Correlation exploitation in error ranking. In *Proceedings of the 12th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESE/FSE)*. ACM, California, 83–93.

[50] Ted Kremenek and Dawson Engler. 2003. Z-Ranking: using statistical analysis to counter the impact of static analysis approximations. In *Proceedings of the 10th International Conference on Static Analysis (SAS)*. ACM, California, 295–315.

[51] Seongmin Lee, Shin Hong, Jungbae Yi, Taeksu Kim, Chul-Joo Kim, and Shin Yoo. 2019. Classifying false positive static checker alarms in continuous integration using convolutional neural networks. In *Proceedings of the 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, Xi'an, 391–401.

[52] Elizaveta Levina and Peter Bickel. 2004. Maximum likelihood estimation of intrinsic dimension. *Advances in Neural Information Processing Systems* 17 (2004), 1–8.

[53] Junjie Li. 2021. A better approach to track the evolution of static code warnings. In *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE/ACM, Madrid, 135–137.

[54] Kaixuan Li, Sen Chen, Lingling Fan, Ruitao Feng, Han Liu, Chengwei Liu, Yang Liu, and Yixiang Chen. 2024. Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Brazil, 921–933.

[55] Kaituo Li, Christoph Reichenbach, Christoph Csallner, and Yannis Smaragdakis. 2014. Residual investigation: Predictive and precise bug detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 1–32.

[56] Guangtai Liang, Ling Wu, Qian Wu, Qianxiang Wang, Tao Xie, and Hong Mei. 2010. Automatic construction of an effective training set for prioritizing static analysis warnings. In *Proceedings of the 25th International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, Antwerp, 93–102.

[57] Kui Liu, Dongsun Kim, Tegawendé F. Bissyandé, Shin Yoo, and Yves Le Traon. 2021. Mining fix patterns for findBugs violations. *IEEE Transactions on Software Engineering (TSE)* 47, 1 (2021), 165–188.

[58] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. *ACM Computing Surveys (CSUR)* 55, 9 (2023), 1–35.

[59] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.11692* (2019), 1–13.

[60] Wei Ma, Shangqing Liu, Menjie Zhao, Xiaofei Xie, Wenhan Wang, Qiang Hu, Jie Zhang, and Yang Liu. 2024. Unveiling Code Pre-Trained Models: Investigating Syntax and Semantics Capacities. *ACM Transactions on Software Engineering Methodology (TOSEM)* (2024). Just Accepted.

[61] Ravi Mangal, Xin Zhang, Aditya V. Nori, and Mayur Naik. 2015. A user-guided approach to program analysis. In *Proceedings of the 10th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Bergamo, 462–473.

[62] Ibéria Medeiros, Nuno F. Neves, and Miguel Correia. 2014. Automatic detection and correction of web application vulnerabilities using data mining to predict false positives. In *Proceedings of the 23rd International Conference on World Wide Web (WWW)*. ACM, Seoul, 63–74.

[63] Qingkun Meng, Chao Feng, Bin Zhang, Chaojing Tang, et al. 2017. Assisting in auditing of buffer overflow vulnerabilities via machine learning. *Mathematical Problems in Engineering* 2017 (2017), 1–14.

[64] Ana Milanova, Atanas Rountev, and Barbara G. Ryder. 2005. Parameterized object sensitivity for points-to analysis for java. *ACM Transaction Software Engineering Methodology (TOSEM)* 14, 1 (jan 2005), 1–41.

[65] Boris Motik and Ulrike Sattler. 2006. A comparison of reasoning techniques for querying large description logic aboxes. In *Proceedings of the 11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*. Springer, Cambodia, 227–241.

[66] Tukaram Muske and Alexander Serebrenik. 2016. Survey of approaches for handling static analysis alarms. In *Proceedings of the 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, North Carolina, 157–166.

[67] Tukaram Muske and Alexander Serebrenik. 2022. Survey of approaches for postprocessing of static analysis alarms. *ACM Computing Survey (CSUR)* 55, 3 (2022), 1–39.

[68] Tukaram B Muske, Ankit Baid, and Tushar Sanas. 2013. Review efforts reduction by partitioning of static analysis warnings. In *Proceedings of the 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. ACM, Eindhoven, 106–115.

[69] Jaechang Nam and Sunghun Kim. 2015. Clami: Defect prediction on unlabeled datasets. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, USA, 452–463.

[70] Kien-Tuan Ngo, Dinh-Truong Do, Thu-Trang Nguyen, and Hieu Dinh Vo. 2021. Ranking warnings of static analysis tools using representation learning. In *Proceedings of the 28th Asia-Pacific Software Engineering Conference (APSEC)*. Taibei, IEEE, 327–337.

[71] Paulo Nunes, Ibéria Medeiros, José Fonseca, Nuno Neves, Miguel Correia, and Marco Vieira. 2017. On combining diverse static analysis tools for web security: an empirical study. In *Proceedings of 13th European Dependable Computing Conference (EDCC)*. ACM, Pisa, 121–128.

[72] Paulo Nunes, Ibéria Medeiros, José C. Fonseca, Nuno Neves, Miguel Correia, and Marco Vieira. 2018. Benchmarking static analysis tools for web security. *IEEE Transactions on Reliability (TR)* 67, 3 (2018), 1159–1175.

[73] Hemant D. Pande, William A Landi, and Barbara G. Ryder. 1994. Interprocedural def-use associations for c systems with single level pointers. *IEEE Transactions on Software Engineering (TSE)* 20, 5 (1994), 385–403.

[74] Sebastiano Panichella, Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. 2015. Would static analysis tools help developers with code reviews?. In *Proceeding of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, Montreal, 161–170.

[75] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, USA, 729–746.

[76] Jose D'Abruzzo Pereira, João R. Campos, and Marco Vieira. 2019. An exploratory study on machine learning to combine security vulnerability alerts from static analysis tools. In *Proceedings of the 9th Latin-American Symposium on Dependable Computing (LADC)*. IEEE, Fortaleza, 1–10.

[77] José D'Abruzzo Pereira, João R Campos, and Marco Vieira. 2021. Machine learning to combine static analysis alerts with software metrics to detect security vulnerabilities: An empirical study. In *Proceedings of the 2021 17th European Dependable Computing Conference (EDCC)*. ACM, Pisa, 1–8.

[78] Meiyuan Qian, Jun Luo, Yu Ge, Chen Sun, Xiuting Ge, and Wanmin Huang. 2021. Semantic-based false alarm detection approach via machine learning. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, Guangzhou, 60–66.

[79] Mingshuang Qing, Xiang Feng, Jun Luo, Wanmin Huang, Jingui Zhang, Ping Wang, Yong Fan, Xiuting Ge, and Ya Pan. 2021. A machine learning-based static analysis warning prioritization. In *Proceedings of the 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, Guangzhou, 685–690.

[80] Mukund Raghothaman, Sulekha Kulkarni, Kihong Heo, and Mayur Naik. 2018. User-guided program reasoning using bayesian inference. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, Pennsylvania, 722–735.

[81] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. 2014. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering (TSE)* 40, 4 (2014), 366–380.

[82] Athos Ribeiro, Paulo Meirelles, Nelson Lago, and Fabio Kon. 2018. Ranking source code static analysis warnings for continuous monitoring of floss repositories. In *Open Source Systems: Enterprise Software and Solutions*. Springer, Athens, 90–101.

[83] Athos Ribeiro, Paulo Meirelles, Nelson Lago, and Fabio Kon. 2019. Ranking warnings from multiple source code static analyzers via ensemble learning. In *Proceedings of the 15th International Symposium on Open Collaboration (OpenSym)*. ACM, Skövde, 10 pages.

[84] Henry Gordon Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Journal of Symbolic Logic* 74, 2 (1953), 358–366.

[85] Joseph R. Ruthruff, John Penix, J. David Morgenthaler, Sebastian Elbaum, and Gregg Rothermel. 2008. Predicting accurate and actionable static analysis warnings: an experimental approach. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Leipzig, 341–350.

[86] Bushra Sabir, Faheem Ullah, M. Ali Babar, and Raj Gaire. 2021. Machine learning for detecting data exfiltration: a review. *ACM Computing Survey (CSUR)* 54, 3 (2021), 1–47.

[87] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. 2018. Lessons from building static analysis tools at Google. *Communications of the ACM,* 61, 4 (2018), 58–66.

[88] Lwin Khin Shar and Hee Beng Kuan Tan. 2012. Mining input sanitization patterns for predicting sql injection and cross site scripting vulnerabilities. In *Proceedins of the 34th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Zurich, 1293–1296.

[89] Haihao Shen, Jianhong Fang, and Jianjun Zhao. 2011. EFindBugs: effective error ranking for findBugs. In *Proceedins of the 4th IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, Valencia, 299–308.

[90] Miltiadis Siavvas, Ilias Kalouptsoglou, Dimitrios Tsoukalas, and Dionysios Kehagias. 2021. A self-adaptive approach for assessing the criticality of security-related static analysis alerts. In *Proceedings of the 21st Computational Science and Its Applications (ICCSA)*. Springer, Cagliari, 289—-305.

[91] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESE/FSE)*. ACM, USA, 1433–1443.

[92] Kien T. Tran and Hieu Dinh Vo. 2022. SCAR: smart contract alarm ranking. In *Proceedings of the 29th Asia-Pacific Software Engineering Conference (APSEC)*. ACM, Japan, 447–451.

[93] Omer Tripp, Salvatore Guarnieri, Marco Pistoia, and Aleksandr Aravkin. 2014. ALETHEIA: improving the usability of static security analysis. In *Proceedings of the 22rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, Denver Colorado, 762–774.

[94] Huy Tu and Tim Menzies. 2021. FRUGAL: Unlocking semi-supervised learning for software analytics. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, Chicago, 394–406.

[95] Yuki Ueda, Takashi Ishio, and Kenichi Matsumoto. 2021. Automatically customizing static analysis tools to coding rules really followed by developers. In *Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Honolulu, 541–545.

[96] Akshay Utture, Shuyang Liu, Christian Gram Kalhauge, and Jens Palsberg. 2022. Striking a balance: pruning false-positives from static call graphs. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Pittsburgh, 2043–2055.

[97] John Viega, J. T. Bloch, Tadayoshi Kohno, and Gary McGraw. 2002. Token-based scanning of source code for security problems. *ACM Transactions on Information and System Security* 5, 3 (2002), 238–261.

[98] Thanh Trong Vu and Hieu Dinh Vo. 2022. Using multiple code representations to prioritize static analysis warnings. In *Proceedings of the 14th International Conference on Knowledge and Systems Engineering (KSE)*. IEEE, Nha Tran, 1–6.

[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 International Conference on Software Engineering (ICSE)*. IEEE/ACM, Pittsburgh, 2377–2388.

[100] Han Wang, Min Zhou, Xi Cheng, Guang Chen, Ming Gu, Lei Bu, and Yingfei Xiong. 2018. Which defect should be fixed first? semantic prioritization of static analysis report. In *Software Analysis, Testing, and Evolution (SATE)*. Springer, Shenzhen, 3–19.

[101] Junjie Wang, Yuchao Huang, Song Wang, and Qing Wang. 2022. Find bugs in static bug finders. In *Proceedings of the 30th International Conference on Program Comprehension (ICPC)*. IEEE/ACM, Online, 516–527.

[102] Junjie Wang, Song Wang, and Qing Wang. 2018. Is there a "golden" feature set for static warning identification? an experimental evaluation. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Oulu, Article 17, 10 pages.

[103] Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2017. OASIS: prioritizing static analysis warnings for android apps based on app user reviews. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, New York, 672–682.

[104] Mark Weiser. 1984. Program slicing. *IEEE Transactions on Software Engineering (TSE)* 4 (1984), 352–357.

[105] Yu Yu Win and Naw Lay Wah. 2023. Actionable static analysis code warnings identification with smote-based classification. In *Proceedings of the 21st IEEE Conference on Computer Applications (ICCA)*. IEEE, Yango, 44–49.

[106] 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, Copenhage, 1–10.

[107] Ge Xiuting, Fang Chunrong, Liu Jia, Qing Mingshuang, Li Xuanye, and Zhao Zhihong. 2023. An unsupervised feature selection approach for actionable warning identification. *Expert Systems with Applications* 227 (2023), 120152.

[108] Xiuting et al. 2024. Reposiroty for my survey. https://github.com/xiaomoqi123/AWISurvey.

[109] Xueqi Yang, Jianfeng Chen, Rahul Yedida, Zhe Yu, and Tim Menzies. 2021. Learning to recognize actionable static code warnings (is intrinsically easy). *Empirical Software Engineering (EMSE)* 26 (2021), 1–24.

[110] Xueqi Yang, Zhe Yu, Junjie Wang, and Tim Menzies. 2021. Understanding static code warnings: an incremental ai approach. *Expert Systems with Applications* 167 (2021), 114134.

[111] Zhao Hong Yang, Yun Zhan Gong, Qing Xiao, and Ya Wen Wang. 2008. DTS - a software defects testing system. In *Proceedings of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, Limassol, 269–270.

[112] Rahul Yedida, Hong Jin Kang, Huy Tu, Xueqi Yang, David Lo, and Tim Menzies. 2023. How to find actionable static analysis warnings: a case study with findBugs. *IEEE Transactions on Software Engineering (TSE)* 49, 4 (2023), 2856–2872.

[113] Sai Yerramreddy, Austin Mordahl, Ugur Koc, Shiyi Wei, Jeffrey S Foster, Marine Carpuat, and Adam A Porter. 2023. An empirical assessment of machine learning approaches for triaging reports of static analysis tools. *Empirical Software Engineering (EMSE)* 28, 2 (2023), 28.

[114] Kwangkeun Yi, Hosik Choi, Jaehwang Kim, and Yongdai Kim. 2007. An empirical study on classification methods for alarms from a bug-finding static c analyzer. *Information Processing letters* 102 (2007), 118–123.

[115] Jongwon Yoon, Minsik Jin, and Yungbum Jung. 2014. Reducing false alarms from an industrial-strength static analyzer by svm. In *Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC)*, Vol. 2. IEEE, Jeju, 3–6.

[116] Ping Yu, Yijian Wu, Xin Peng, Hahjia Peng, Jian Zhang, Peicheng Xie, and Wenyun Zhao. 2023. ViolationTracker: building precise histories for static analysis violations. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. IEEE/ACM, Melbourne, 1–12.

[117] Ulas Yüksel and Hasan Sözer. 2013. Automated classification of static code analysis alerts: a case study. In *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, Eindhoven, 532–535.

[118] Ulaş Yüksel, Hasan Sözer, and Murat Şensoy. 2014. Trust-based fusion of classifiers for static code analysis. In *Proceedings of the 17th International Conference on Information Fusion (FUSION)*. IEEE, South Carolina, 1–6.

[119] Huaien Zhang, Yu Pei, Junjie Chen, and Shin Hwei Tan. 2023. Statfier: automated testing of static analyzers via semantic-preserving program transformations. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, New York, 237–249.

[120] Xin Zhang, Xujie Si, and Mayur Naik. 2017. Combining the logical and the probabilistic in program analysis. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*. ACM, Barcelona, 27–34.

[121] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1 (2010), 43–52.

[122] Yuwei Zhang, Ying Xing, Yunzhan Gong, Dahai Jin, Honghui Li, and Feng Liu. 2020. A variable-level automated defect identification model based on machine learning. *Soft Computing* 24 (2020), 1045–1061.

[123] Yunhui Zheng, Saurabh Pujar, Burn Lewis, Luca Buratti, Edward Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. 2021. D2A: a dataset built for ai-based vulnerability detection methods using differential analysis. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, Madrid, 111–120.

Received 29 November 2023; revised 24 August 2024; accepted 12 September 2024