

CSVD-AES: Cross-project software vulnerability detection based on active learning with metric fusion

Zhidan Yuan^{a,b}, Xiang Chen^c, Juan Zhang^b, Weiming Zeng^{a,*}

^a College of Information Engineering, Shanghai Maritime University, Shanghai, China

^b College of Information Engineering, Jiangsu Maritime Institute, Nanjing, China

^c School of Artificial Intelligence and Computer Science, Nantong University, Nantong, China

ARTICLE INFO

Keywords:

Cross-project software vulnerability detection

Active learning

Metric fusion

Class imbalance

ABSTRACT

Context: Previous studies on Cross-Project Software Vulnerability Detection (CSVD) have shown that leveraging a small number of labeled modules from the target project can enhance the performance of CSVD. However, how to systematically select representative modules for labeling has not received sufficient attention. In addition, program modules can be measured using either expert or semantic metrics. There has been insufficient attention given to whether considering both metrics simultaneously helps in selecting representative modules.

Objective: To address these challenges, we introduce a novel approach CSVD-AES. This method aims to fuse expert and semantic metrics and employs the active learning to select the most representative modules for labeling.

Methods: CSVD-AES consists of three phases: the code representation phase, the active learning phase, and the model construction phase. In the code representation phase, a self-attention mechanism is used to fuse the metrics. In the active learning phase, an uncertainty sampling strategy is employed to select the most representative modules for labeling. In the model construction phase, the weighted cross-entropy (WCE) loss function is applied to address the class imbalance issue in the labeled modules. The metric fusion helps active learning identify representative modules. Since selecting modules can exacerbate the class imbalance issue in the labeled modules, we employ a sampling balancing strategy during the active learning phase to address this problem.

Results: CSVD-AES is evaluated through a comprehensive study on four real-world projects. The results demonstrate that CSVD-AES outperforms five state-of-the-art baselines, achieving AUC improvements ranging from 4.0% to 24.4%. A series of ablation experiments verify the rationality of the CSVD-AES component settings.

Conclusion: CSVD-AES effectively addresses the challenges in the field of CSVD by combining active learning and metric fusion, significantly advancing the development of this field.

1. Introduction

Security vulnerabilities in source code have become a significant threat to modern software development [1]. Malicious attackers can exploit these flaws to compromise system confidentiality, integrity, and availability. For instance, in 2021, attackers leveraged the ProxyLogon vulnerability [2] to steal Acer's internal data and demanded a \$50 million ransom. Consequently, developing effective software vulnerability detection (SVD) methods is essential.

Recent advances in deep learning have led to the development of numerous vulnerability detection methods [3–8]. These approaches automatically extract features from code to train or fine-tune vulnerability detection models, with most relying on transformer or graph neural networks, which outperform traditional methods (such as program analysis). While recent studies have explored Large Language Model (LLM) based detection techniques [9,10], Yin et al. [11] found that fine-tuned LLMs perform weaker than Transformer-based methods. Given the

* Corresponding author.

E-mail addresses: yuanzhidan@stu.shmtu.edu.cn (Z. Yuan), xchencs@ntu.edu.cn (X. Chen), 20060529@jmi.edu.cn (J. Zhang), wmzeng@shmtu.edu.cn (W. Zeng).

computational and time costs of deploying LLMs, Transformer-based approaches remain the most efficient option.

Most deep learning-based SVD studies rely on datasets from the same software project. However, obtaining high-quality labeled SVD datasets for the target project is challenging. Labeling source code is time-consuming, labor-intensive, and error-prone, even for security experts [12–14]. To tackle this issue, Cross-project Software Vulnerability Detection (CSVD) methods have been introduced. These methods leverage labeled SVD datasets from source projects to build models that predict vulnerabilities in target projects. The source project contains labeled program modules, whereas the target project has unlabeled modules. However, due to the distribution differences between the source and target projects, the performance of CSVD models is often suboptimal. Previous studies have employed transfer learning methods to address this issue. Refs. [15–19] adopt domain adaptation to reduce the distribution gap between the source and target projects, while Ref. [20] leverages TrAdaBoost [21] combined with expert and semantic metrics to further mitigate this problem. TrAdaBoost is a representative semi-supervised transfer learning approach. Building on this method, Cai et al. [20] proposed enhancements to further alleviate the distribution shift between source and target projects. Moreover, they introduced a novel setting in which a small subset of program modules from the target project is selected and incorporated into the source project. This setting can be viewed as a form of semi-cross-project software vulnerability detection, bridging the gap between traditional within-project and fully cross-project scenarios.

However, previous research on CSVD still faces three key limitations in practical applications. First, existing study [20] has not sufficiently investigated systematic strategies for selecting representative modules for labeling. Second, it remains uncertain whether the joint consideration of expert metrics and semantic metrics can enhance the selection of representative program modules in the target project. Third, current research lacks in-depth exploration of effective strategies to mitigate the impact of class imbalance during the module selection process. To address the three limitations mentioned above, we propose a novel approach CSVD-AES (Cross-project Software Vulnerability Detection based on Active learning by fusing Expert metrics and Semantic metrics).

To alleviate the first limitation, we adopt the active learning [22] to make the selection of program modules more strategic. To alleviate the second limitation, we employ a self-attention mechanism [23] to integrate expert metrics and semantic metrics. To mitigate the third limitation, we propose a sampling balancing strategy to balance the selected modules from the target project during the selection phase. Furthermore, we employ the Weighted Cross-Entropy (WCE) loss [24] during model training to further alleviate the impact of class imbalance.

To verify the effectiveness of our proposed CSVD-AES, we perform extensive experiments on four practical software projects [25]: FFmpeg, LibPNG, LibTIFF, and VLC. The experimental results clearly show that, in terms of AUC performance, CSVD-AES surpasses the state-of-the-art baselines. Its AUC value is 0.811, achieving a performance improvement of 4.0% to 24.4% compared to the baselines. The ablation study results demonstrate that the component settings of CSVD-AES are all effective. Specifically, CSVD-AES improves the AUC score by 5.6% and 33.8% compared to CSVD-AES without the sampling balancing strategy and CSVD-AES without active learning. Compared to five other active learning strategies, the uncertainty sampling strategy we use improves the AUC score by 1.8% to 9.6%. In CSVD-AES, we sample 30% of the program modules from the target project for labeling. Empirical results show that sampling 30% of the program modules outperforms sampling 10%, 20%, and 40%. Compared to using only expert metrics, only semantic metrics, and only concatenating both types of metrics, CSVD-AES improves the AUC score by 15.0%, 6.0%, and 2.8%, respectively. Compared to using cross-entropy loss and focal loss, CSVD-AES improves the AUC score by 5.2% and 7.0%, respectively.

Impact of study. We propose a novel CSVD method CSVD-AES. CSVD-AES addresses the challenge of how to systematically select representative modules from the target project for labeling using active learning and further improves the performance of the CSVD model through metric fusion. Additionally, we mitigate the impact of class imbalance in CSVD from both the loss function and active learning perspectives. To facilitate and encourage future work in CSVD, we share our source code and dataset in the GitHub repository (<https://github.com/yzdAtom/CSVD-AES>).

The originality and contributions of our research are as follows:

- **Active learning for selecting representative modules in the target project.** To the best of our knowledge, we are the first to apply active learning methods to CSVD. We use an uncertainty sampling strategy to select the most representative modules from the target project for labeling and add them to the source project. To address the issue that selecting modules from the target project may exacerbate the class imbalance problem, we propose a sampling balancing strategy. This strategy improves the active learning method to mitigate the adverse effects of class imbalance.
- **A novel modeling approach that fuses expert metrics and semantic metrics.** Different from previous model fusion methods [20], we design a novel CSVD model that integrates expert and semantic metrics using a self-attention mechanism, without the need for manually adjusting the model weights of different metrics. Additionally, we use a weighted cross-entropy (WCE) loss function for model training to handle the class imbalance issue.
- **Comprehensive experimental evaluation.** We conduct experiments on four real-world software projects. To mitigate the influence of random factors, we perform multiple experimental rounds and conduct statistical analyses on all methods. The results demonstrate that CSVD-AES outperforms five state-of-the-art baseline methods in the CSVD scenario. Additionally, we evaluate the influence of various active learning strategies, the use of a single metric, and different loss functions on the performance of CSVD-AES. The findings confirm the effectiveness of the module configurations within CSVD-AES.

Paper Organization. In Section 2, we present the background and motivation of this study. Section 3 offers a detailed account of the overall framework of CSVD-AES along with the specifics of each phase. Section 4 details our experimental setup. Section 5 analyzes the experimental results for each research question and provides a summary. Section 6 explores potential threats to our study. Section 7 reviews related work and emphasizes the novelty of our research. Finally, Section 8 provides a conclusion of our work and shows potential future directions.

2. Background

In this section, we first provide an overview of the background of CSVD, followed by an analysis of the limitations in previous CSVD research.

2.1. Cross-project software vulnerability detection

Although extensive research has been conducted on deep learning-based SVD, in practical applications, target projects are often unlabeled. Labeling the program modules within a project is both time-consuming and tedious, and even experienced security experts can make mistakes [12].

Cross-project software vulnerability detection (CSVD) aims to utilize labeled program modules from other projects (source projects) to build the SVD model, which can identify vulnerabilities in the current project (target project). This method helps reduce the substantial costs

associated with labeling program modules. However, because of the disparities in data distribution between the source project and the target project, the CSVD model frequently fails to achieve optimal performance. Previous studies [15–20] have primarily focused on addressing the data distribution differences in CSVD from the perspective of transfer learning. Specifically, these studies have utilized techniques such as domain adaptation [26] and TrAdaBoost [21] to mitigate this issue.

2.2. Research motivation

Most previous studies [15–20] have achieved promising results from the perspective of transfer learning. However, they still have three limitations.

Limitation ①: Previous studies [20] have found that labeling a small number of program modules from the target project helps improve the performance of CSVD. However, these studies have overlooked the process of selecting the modules from the target project. They often rely on random sampling without considering its impact. This lack of strategic selection can significantly influence model performance. How to systematically select representative modules for labeling has not received sufficient attention in existing research.

Limitation ②: The metrics for program modules can be measured using either expert metrics or semantic metrics. It is still unclear whether considering both types of metrics simultaneously helps in selecting representative program modules from the target project. Although previous studies [20] have attempted to fuse expert metrics and semantic metrics, they did not consider the impact of metric fusion on module selection. Additionally, there are some drawbacks in the previous methods. They use traditional machine learning models like XGBoost [27] for model fusion, which required manual adjustment of the weights for models trained on the two types of metrics (e.g., Cai et al. [20] use weights of 0.4 and 0.6). Moreover, semantic metrics derived from Pre-trained Language Models (PLMs) with default parameters fail to adequately capture the semantic structure of the code. Fine-tuning these models using existing SVD data can yield better code representations.

Limitation ③: Class imbalance is a common issue in SVD. In the CSVD scenario, selecting program modules from the target project for labeling may further exacerbate the class imbalance problem in the labeled modules, as there is often a large number of non-vulnerable modules in the target project. How to mitigate the impact of class imbalance during the module selection process has not received sufficient attention. Since previous research [16–18] rarely considered the scenario of selecting program modules for labeling from the target project, most studies still addressed the class imbalance problem starting from the entire training set. Cai et al. [20] applied the classic SMOTE method [28] to generate samples for the minority class, but this method is not suitable for fine-tuning PLMs. As training progresses, the semantic features derived from PLMs are not fixed. Nguyen et al. [16, 17] argued that deep domain adaptation can indirectly address the class imbalance issue. They further used the Max-Margin Principle and Random Feature Map to solve this problem [18]. However, the Max-Margin Principle has high computational complexity and assumes that data is linearly separable in the feature space. In practice, software vulnerability and non-vulnerability modules are often not linearly separable, which may cause models to fail to achieve the expected performance.

3. Approach

This section introduces the framework of CSVD-AES and details its key components.

3.1. Framework of CSVD-AES

In traditional CSVD methods, labeled modules from the target project are typically unavailable during the training phase. However, as is commonly practiced in cross-project defect prediction [29–32], incorporating a small portion of labeled modules from the target project is acceptable and does not compromise the overall cross-project setting. This forms what we call a semi-cross-project detection scenario. CSVD-AES adopts this setting by iteratively selecting a limited number of target-project program modules for expert labeling, thereby bridging the gap between conventional within-project detection and fully cross-project detection.

Fig. 1 shows the framework of CSVD-AES and its pseudocode can be found in Algorithm 1. CSVD-AES is a CSVD method that integrates active learning and metric fusion. This approach consists of three key phases: code representation phase, active learning phase, and model construction phase. In the code representation phase, we extract both expert (Line 6) and semantic metrics (Line 7), where the representation of semantic metrics evolves dynamically as the model is trained (Line 12). The expert and semantic metrics are then concatenated (Line 9) and fused using a self-attention mechanism (Line 10), producing the final code representation. In the active learning phase, we adopt an uncertainty sampling strategy (Line 13) to select the most representative program modules from the target project for expert labeling (Line 14). Once the experts have labeled the modules, a sampling balancing strategy (Line 16) is applied to balance all labeled modules. The processed modules are then added to the source project (Line 17) for continued training. During the model construction phase, the code representation is fed into a single-layer fully connected network (Line 11). To address the class imbalance in the SVD dataset, we employ the WCE loss function, which assigns greater importance to the minority class. The complete model includes the code representation component, and when updating model parameters, the parameters of this component are updated simultaneously (Line 12). These three phases are repeated iteratively until the training process is complete (Line 5).

3.2. Code representation phase

In this phase, our objective is to transform code into computable numerical forms. For CSVD-AES, we derive code representations from two aspects: expert metrics and semantic metrics. A self-attention mechanism is then employed to fuse these metrics, producing the final code representation. Notably, unlike in previous study [20], the code representation dynamically evolves as the model parameters are updated.

3.2.1. Expert metrics extraction

In our research, we adopt the same 39 expert metrics as those used in Cai et al.'s study [20]. Previous research [20] has confirmed the effectiveness of these metrics, showing no significant feature redundancy. We use the commercial tool Understand¹ to extract code metrics. Understand is a static code analysis tool that computes code metrics for software projects written in Java, C/C++, Python and other programming languages. It is developed by SciTools. These 39 expert metrics are categorized into five dimensions: performance, maintainability, code size, readability, and complexity.

Specifically, performance metrics [33] evaluate the efficiency and speed of software execution. Although the correlation is indirect, performance issues may reflect inefficient or flawed implementation logic, which in turn increases the likelihood of introducing security vulnerabilities. Maintainability [34] refers to the ease with which software can be modified while minimizing the risk of introducing new defects. This aspect can be quantified using the number of statements in the

¹ <http://www.scitools.com/>

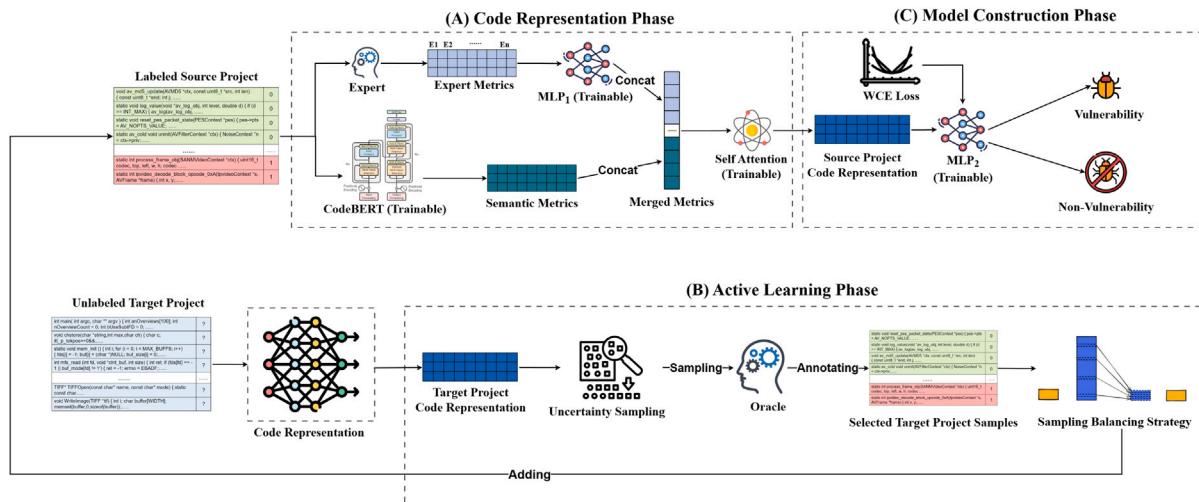


Fig. 1. The framework of our proposed approach CSVD-AES.

Algorithm 1: Pseudocode for CSVD-AES**Input:**

Labeled functions in the source project: S
 Unlabeled functions in the target project: T
 Sampling percentage: sp
 Total training epochs: E

Output:

Trained CodeBERT model: $CodeBERT$
 Trained MLP model: MLP_1, MLP_2
 1: $ALNum \leftarrow sp \times size(T)$
 2: $sn \leftarrow ALNum/E$
 3: $Allselected \leftarrow \emptyset$
 4: $S_{ori} \leftarrow S$
 5: **for** $epoch=1$; $epoch \leq E$; $epoch++$ **do**
 6: $EM \leftarrow Understand(S)$
 7: $SM \leftarrow CodeBERT(S)$
 8: $EM \leftarrow MLP_1(EM)$
 9: $CR \leftarrow Concat(SM, EM)$
 10: $CR \leftarrow Self\ Attention(CR)$
 11: $PL \leftarrow MLP_2(CR)$
 12: train $CodeBERT, MLP_1, MLP_2$
 13: $selected \leftarrow ALSampling(T, sn)$
 14: assign labels to $selected$ via Experts
 15: $Allselected \leftarrow Allselected \cup selected$
 16: $selected2 \leftarrow BalanceSample(Allselected)$
 17: $S \leftarrow S_{ori} \cup selected2$
 18: $T \leftarrow T \setminus selected$
 19: **end for**
 20: **return** $CodeBERT, MLP_1, MLP_2$

application. In general, better maintainability contributes to a lower risk of security vulnerabilities. Code size [35] is commonly used to estimate the effort required for software development. It provides an indication of programming productivity and maintainability, and is typically measured using metrics related to lines of code. Readability [36] represents the perceived ease with which developers can understand the source code. Existing research [37] has demonstrated a significant correlation between readability and software vulnerabilities. Code segments with lower readability are more likely to contain security flaws. Complexity [38] captures the difficulty of comprehending the logic, structure, and organization of a program. It is affected by various factors, such as the depth of nesting and inheritance, as well as the use

of sophisticated algorithms and data structures. High complexity can hinder error correction, feature enhancement, and code modification, increasing the risk of unintended consequences and the introduction of new vulnerabilities.

Table 1 provides detailed information on these expert metrics, including their names, descriptions, and dimensions.

3.2.2. Semantic metrics extraction

In this study, we utilize CodeBERT [39] to derive semantic metrics from program modules. Fundamentally, semantic metrics are context-aware word embeddings that capture the contextual semantics of program elements. Many prior studies [40–43] have leveraged this model to learn context-aware word embeddings and have achieved promising results. CodeBERT adopts the BERT [44] architecture, consisting of 12 Transformer [23] blocks, with a hidden layer dimension of 768. Specifically, for a function-level code snippet, we first split it using the camel case naming convention to get the code sequence. Then, we feed the code sequence into CodeBERT to obtain the sequence vector representation s_x for each source code function, capturing both global and local information. The computation process is as follows:

$$s_x = L^0[CLS] + L^N[CLS] \quad (1)$$

where L^0 and L^N represent the embeddings from the first and last layers of CodeBERT, respectively. “[CLS]” is a special token that is typically used to represent the semantic information of the entire sequence. During model training, we do not freeze the parameters of CodeBERT. Instead, we fine-tune CodeBERT using labeled function-level program modules. This allows CodeBERT to better capture the code semantic features relevant to the current project. Therefore, s_x is not fixed during training. It changes as the parameters of the CodeBERT are updated.

3.2.3. Metric fusion method

Through expert and semantic metric extraction, we obtain the expert metric representation ex and the semantic metric representation s_x for each function-level code snippet. We define EM ($EM = \{ex_i, i \in \{1, 2, \dots, n\}\}$) as the set of expert metrics for all labeled modules and SM ($SM = \{s_x, i \in \{1, 2, \dots, n\}\}$) as the set of semantic metrics for all labeled modules. n represents the number of all labeled modules. To integrate these two metric representations, we first concatenate them column-wise. Since EM contains 39 numeric elements and SM has 768 numeric elements, directly concatenating EM and SM would cause SM to dominate the model's influence. To address this issue, we follow a similar approach to Ni et al. [45] by applying a fully connected

Table 1

Explanation of expert metrics utilized by CSVD-AES.

Dimension	Name	Description
Performance	ExecutableLineCount	Quantity of lines in the executable source code
	SemicolonCount	Number of semicolons present
Maintainability	StatementCount	Total number of statements
	DeclarativeStmtCount	Count of declarative statements
	EmptyStmtCount	Quantity of empty statements
	ExecStmtCount	Quantity of executable statements
Code size	ClassCount	Number of defined classes
	FunctionCount	Total number of functions
	TotalLineCount	Number of lines
	BlankLineCount	Quantity of blank lines (BLOC)
	SourceLineCount	Quantity of source code lines (LOC)
	DeclSourceLineCount	Number of declarative source code lines
	CommentLineCount	Count of comment lines (CLOC)
	InactiveLineCount	Quantity of inactive lines
	PreprocessorLineCount	Quantity of lines with preprocessor directives
Readability	AvgLineFunc	Average lines in all functions/methods nested within others
	AvgBlankLine	Mean number of blank lines in all inner functions/methods
	AvgSourceLine	Average line count of source code in all nested functions/methods
	AvgCommentLine	Average number of comment lines within all nested functions/methods
	AltAvgBlankLine	Average count of blank lines in all inner functions/methods (including inactive areas)
	AltAvgSourceLine	Average line count of source code within all inner functions or methods (including inactive areas)
	AltAvgCommentLine	Average count of comment lines in all inner functions or methods (including inactive areas)
	AltBlankLineCount	Number of empty lines, considering inactive sections
	AltSourceLineCount	Number of code lines, considering inactive parts
	AltCommentLineCount	Number of comment lines, considering inactive parts
Complexity	CommentToCodeRatio	Percentage of comment lines relative to code lines
	AvgCyclomaticComplexity	Average cyclomatic complexity across all inner functions/methods
	AvgModifiedCyclomatic	Average adjusted cyclomatic complexity across all nested functions/methods
	AvgStrictCyclomatic	Average value of strict cyclomatic complexity for all inner functions/methods
	AvgEssentialComplexity	Average core complexity across all nested functions/methods
	MaxCyclomaticComplexity	Highest cyclomatic complexity among all nested functions/methods
	MaxModifiedCyclomatic	Highest modified cyclomatic complexity of nested functions/methods
	MaxStrictCyclomatic	Highest strict cyclomatic complexity of nested functions/methods
	MaxEssentialComplexity	Highest essential complexity among all nested functions/methods
	MaxNestingLevel	Maximum depth of nested control structures
	SumCyclomaticComplexity	Total cyclomatic complexity of all nested functions/methods
	SumModifiedCyclomatic	Total modified cyclomatic complexity of all nested functions/methods
	SumStrictCyclomatic	Total strict cyclomatic complexity of all nested functions/methods
	SumEssentialComplexity	Total essential complexity of all nested functions/methods

layer to expand the dimensionality of EM into a higher-dimensional representation (Line 8). After this process, the dimensionality of EM is increased to 768. The computation process for this method is as follows:

$$EM = W \cdot EM + b, \quad W \in \mathbb{R}^{d_2 \times d_1}, \quad b \in \mathbb{R}^{d_2} \quad (2)$$

where W and b represent the weights and biases of the fully connected layer. d_1 represents the number of numeric elements in EM , and its value is 39. d_2 represents the number of numeric elements in SM , and its value is 768. Subsequently, we concatenate SM and EM column-wise. The computation process for column-wise concatenation is as follows:

$$CR = [SM \parallel EM] \in \mathbb{R}^{n \times 2d_2}, \quad SM, EM \in \mathbb{R}^{n \times d_2} \quad (3)$$

Since not all metric features contribute equally to code representation, we draw inspiration from feature fusion techniques in the image domain [46] and employ a self-attention mechanism [23] to integrate the two types of metrics. The core idea of the self-attention mechanism is to compute the relative importance between elements within the same input sequence, thereby dynamically adjusting the information weights of different parts. By automatically learning the weights of different metrics, the self-attention mechanism can dynamically adjust the contributions of expert metrics and semantic metrics, ensuring a reasonable fusion of both types of information. Specifically, the calculation process of the self-attention mechanism is as follows:

$$Q = CR \times W_Q, \quad K = CR \times W_K, \quad V = CR \times W_V \quad (4)$$

In this context, Q , K , and V represent the query matrix, key matrix, and value matrix, respectively. Next, the similarity is computed by taking the dot product of the query matrix Q and the key matrix K , and the result is scaled by a factor d (where $d = 1536$ in our study). Then, the softmax function is applied to each row for normalization, resulting in the attention weight matrix A . A represents the relative importance between the samples. Finally, the attention weight matrix A is used to perform a weighted sum of the value matrix V , yielding the final output representation. The detailed computation process is as follows:

$$A = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right), \quad CR = AV \quad (5)$$

The output after metric fusion has the same dimension as the input.

3.3. Active learning phase

In this phase, our goal is to use active learning method to select the most representative program modules from the target project for labeling, and then add them to the source project for model training. We use uncertainty sampling strategy to select the modules. To mitigate the class imbalance issue during the sample selection process, we employ a sampling balancing strategy to balance the sampled modules from the target project.

3.3.1. Uncertainty sampling strategy

In our work, the goal of active learning is to select the most representative program modules from the target project for expert labeling. After the expert labels the modules, they are added to the

source project to contribute to the training of the CSVD model. Various sample selection strategies exist in active learning, such as margin sampling [47], entropy sampling [48], and uncertainty sampling [49]. These strategies differ significantly in how they prioritize the most representative modules for labeling. Among these, uncertainty sampling is one of the most commonly employed methods. In our approach, we simulate the expert labeling process by querying the true labels of program modules. The aim of uncertainty sampling is to select the most uncertain samples from the unlabeled set for expert labeling. In our study, we utilize the uncertainty measurement approach from literature [50] to compute the uncertainty scores of the unlabeled code samples. Specifically, the uncertainty score is measured from two aspects: minimizing the upper bound of the worst-case scenario for a single sample and minimizing the upper bound of the L_1 norm. Minimizing the upper bound of the worst-case scenario aims to identify the samples with the highest minimum predicted confidence score. The specific calculation method is as follows:

$$\text{score}_1 = \min \max_{y_i \in \{0,1\}} -\log(p_{y_i}) \quad (6)$$

For example, suppose we have two code samples. The first sample has $p_0 = 0.4$ and $p_1 = 0.6$. The second sample has $p_0 = 0.3$ and $p_1 = 0.7$. According to Eq. (6), the first sample's $\max_{y_i \in \{0,1\}} -\log(p_{y_i}) = 0.398$. The second sample's $\max_{y_i \in \{0,1\}} -\log(p_{y_i}) = 0.523$. Since the result for the first sample is smaller, we choose the first sample as the query sample.

Minimizing the L_1 norm upper bound represents selecting samples with uniform prediction confidence scores. The calculation process is as follows:

$$\text{score}_2 = \min \sum_{y_i \in \{0,1\}} -\log(p_{y_i}) \quad (7)$$

When the prediction confidence trend of a code sample is more uniform, the sample will be more uncertain. When the output scores are uniform, score_2 will tend to take the minimum value. In this study, the final uncertainty score is the weighted sum of the two components, and the specific calculation method is as follows:

$$us = \min(\alpha \cdot \text{score}_1 + (1 - \alpha) \cdot \text{score}_2) \quad (8)$$

The parameter α is a hyperparameter used to balance the two uncertainty measures. In our study, the value of α is 0.5.

3.3.2. Sampling balancing strategy

Since the program modules in the target project often suffer from class imbalance, the active learning phase tends to select a large number of non-vulnerable modules (label 0). This will worsen the class imbalance issue in the training modules. During model training, if a series of consecutive epochs selects mostly 0-label modules, the model becomes more biased towards predicting the majority class, leading to a decline in performance.

To address this issue, we have designed a sampling balancing strategy. The pseudocode for the sampling balancing strategy is shown in Algorithm 2. In Algorithm 2, we first obtain the number of labeled non-vulnerable modules (Line 1) and vulnerable modules (Line 2) from the target project. When both the number of vulnerable and non-vulnerable modules are non-zero (Line 3), we balance the selected data in two cases. When the number of vulnerable modules is greater than or equal to the number of non-vulnerable modules, we do not make any adjustments and directly return all labeled modules *Allselected* (Line 4–6). When the number of vulnerable modules is less than the number of non-vulnerable modules, we perform random undersampling on *Allselected* (Line 7–9). We randomly remove non-vulnerable modules from *Allselected* until the number of non-vulnerable samples equals the number of vulnerable modules. If there are no non-vulnerable modules in *Allselected* and only vulnerable modules exist, we do not perform any processing on *Allselected* (Line 11–13). If there are no vulnerable

modules in *Allselected* and only non-vulnerable modules exist, we discard all labeled modules and train the model only using the source project (Line 14–16).

In the sampling balancing strategy, if the number of vulnerable modules is greater than non-vulnerable modules, we do not perform any processing. This approach is based on previous research [51], which indicates that vulnerable modules are crucial for model construction, while non-vulnerable modules may contain some noise. Therefore, we never discard the vulnerable modules labeled from the target project under any circumstances.

Algorithm 2: Pseudocode for sampling balancing strategy(SBS)

Input:

All samples selected from T : *Allselected*

Output:

Allselected processed by the SBS: *selected2*

```

1: zero_num ← Size_0(Allselected)
2: one_num ← Size_1(Allselected)
3: if zero_num ≠ 0 and one_num ≠ 0 then
4:   if one_num ≥ zero_num then
5:     selected2 ← Allselected
6:   end if
7:   if one_num < zero_num then
8:     selected2 ← RandomUnderSamp(Allselected)
9:   end if
10: end if
11: if zero_num == 0 and one_num ≠ 0 then
12:   selected2 ← Allselected
13: end if
14: if one_num == 0 and zero_num ≠ 0 then
15:   selected2 ← ∅
16: end if
17: return selected2
```

3.4. Model construction phase

In the model construction phase, we use the output *CR* from the code representation phase as the input for this stage. A single-layer fully connected network is employed as the classification model. Previous SVD studies [6,15] commonly use cross-entropy loss as the loss function during training. However, SVD datasets often suffer from class imbalance, which causes the model to predominantly predict the majority class when trained with cross-entropy loss. To address this issue, prior studies [20] frequently use oversampling techniques, such as the SMOTE method. However, sampling methods can distort the actual distribution of the data. As the model parameters are updated, oversampling strategies become increasingly difficult to apply, and these methods do not adequately resolve the model's reduced ability to distinguish minority class samples.

Therefore, from the perspective of the loss function, we use weighted cross-entropy (WCE) loss [24] to train the CSVD model. This loss function enables the model to pay more attention to the minority vulnerable program modules. The calculation formula for WCE loss is as follows:

$$WCE = -\frac{1}{n} \sum_{i=1}^n [w_{y_i} \cdot \log(p_{y_i})], \quad y_i \in \{0, 1\} \quad (9)$$

In Eq. (9), y_i represents the label of the i th program module, where the value is either 0 or 1. The value of 0 indicates that the module has no vulnerability, and 1 indicates that the module has vulnerabilities. p_{y_i} represents the probability predicted by the model that the i th program module belongs to the label y_i . w_{y_i} represents the class weight. When $y_i = 1$, $w_{y_i} = n_0/n$; when $y_i = 0$, $w_{y_i} = n_1/n$. n_0 represents the number of program modules with no vulnerabilities (label 0), n_1 represents the number of program module with vulnerabilities (label 1), and n represents the total number of labeled modules.

4. Experimental setup

This section presents the research questions (RQs) and their corresponding design motivations.

4.1. Research questions

To show the effectiveness of CSVD-AES and the rationality behind each component of the method, we have designed four RQs in our empirical study.

RQ1: How does our proposed CSVD-AES perform compared to the baselines?

Motivation: In RQ1, we aim to compare the performance of CSVD-AES with baselines to validate its effectiveness. We compare it with three recent CSVD methods (Dual-GD-DDAN [17], DAM2P [18] and CSVD-TF [20]), four vulnerability detection methods that perform well in the within-project scenario (LineVul [6], VulBERT [52], VulBERTa-MLP [53] and VulBERTa-CNN [53]), and a smart contract vulnerability detection method that applies active learning in the within-project scenario (ASSBert [49]). These methods are described in detail in Section 4.3.

RQ2: What impact does the incorporation of active learning have on the performance of CSVD-AES?

Motivation: In RQ2, our goal is to assess how active learning methods influence the performance of CSVD-AES. First, we compare the performance difference between using and not using active learning in the CSVD-AES. Then, we examine the effect of different active learning approaches on the performance of CSVD-AES to verify whether the active learning method we used has advantages. Finally, we explore the impact of the proportion of samples selected from the target project using active learning on the performance of CSVD-AES.

RQ3: Can the performance of CSVD-AES be improved by fusing two types of metrics?

Motivation: In RQ3, we aim to explore whether the fusion of the two metrics is effective for the CSVD-AES. We compare CSVD-AES with methods using only expert metrics and only semantic metrics to verify the effectiveness of fusing both types of metrics. Additionally, we also consider the scenario where expert metrics and semantic metrics are simply concatenated to assess the effectiveness of using the self-attention mechanism.

RQ4: How does the WCE loss function influence the performance of CSVD-AES?

Motivation: In RQ4, we aim to verify whether using the WCE loss can improve the performance of CSVD-AES. The SVD dataset often suffers from class imbalance, which leads to the model being biased toward predicting samples as the majority class. We use the WCE loss to assign weights to the loss for different classes, making the model focus more on vulnerable samples. We will compare CSVD-AES using the WCE loss with CSVD-AES using standard cross-entropy loss and using focal loss [54] to demonstrate the effectiveness of the WCE loss.

4.2. Experimental subjects

We use the dataset provided by Lin et al. [25] in our empirical study. The dataset involves four real-world large-scale projects: FFmpeg, LibPNG, LibTIFF, and VLC. The projects serve as the subjects for the experiments. All four projects are developed in the C programming language. Specifically, FFmpeg is a cross-platform multimedia processing toolkit. LibTIFF is used for handling TIFF files. LibPNG is used for handling PNG format images. VLC is a multimedia player that supports multiple platforms. The National Vulnerability Database² (NVD) and the Common Vulnerabilities and Exposures³ (CVE) provide

Table 2
Statistical information of experimental subjects.

Project	# Vulnerable functions	# Non-vulnerable functions	% Vulnerable functions
FFmpeg	213	5552	3.695%
LibTIFF	96	729	11.636%
LibPNG	44	575	7.108%
VLC	42	6110	0.683%

the vulnerability data. These datasets have been used in previous CSVD studies [18,20], ensuring the validity of the experimental conclusions.

We find that a small number of files in the four projects cannot form complete functions, and some files are empty. We manually removed these data. Table 2 presents the statistical information of our experimental subjects, including the number of vulnerable functions, the number of non-vulnerable functions, and the proportion of vulnerable functions out of the total number of functions. We can observe that all four datasets suffer from class imbalance, with VLC having the most severe imbalance, where vulnerable functions account for less than 1% of the total functions.

4.3. Baselines

We consider eight baselines that are relevant to our research, including three recent CSVD methods and five methods for SVD in the within-project scenario. Since there are two different scenarios in current CSVD methods (i.e., whether some data from the target project is selected and labeled), we need to set up the baselines accordingly. To facilitate the description of the baseline settings, we adopt the same labeling conventions as in Algorithm 1 for the related data. We denote the labeled SVD data from the source project as S , the labeled samples selected from the target project using active learning as $Allselected$, and the test dataset of the target project as T_T . For CSVD-AES, we use S and $Allselected$ processed by the sampling balancing strategy to train the model and evaluate its performance using T_T . The introduction of our baselines is as follows.

CSVD-TF [20]: Cai et al. used CodeBERT to extract semantic metrics and Understand to extract expert metrics. They applied a model fusion approach, training separate XGBoost models for expert and semantic metrics. Weights are then assigned to the two models to obtain the final model. They used TrAdaboost to reduce the distribution discrepancy between the source and target projects. They randomly selected program modules from the target project for labeling to improve the performance of the constructed model. Since this baseline method involves randomly selecting samples from the target project, we use this method to select the same number of samples as $Allselected$ from the target project, denoted as $BLselected$. We train the model using S and $BLselected$, and evaluate the model using T_T .

Dual-GD-DDAN [17]: Nguyen et al. introduced the Dual-GD-DDAN. It consists of two generators and two discriminators, which are used to map data from the source and target domains into a joint feature space and distinguish between source and target projects within this space. This method aims to reduce the distribution discrepancy between the source and target projects while avoiding data degradation and boundary distortion issues. Since this baseline cannot select modules from the target project, we use S and $Allselected$ as training set for fairness. We use T_T as the testing set to assess the performance of the model.

DAM2P [18]: Nguyen et al. introduced the DAM2P. It is a deep domain adaptation method for CSVD. DAM2P first utilizes a Generative Adversarial Network framework to map the source code data into a joint feature space, making the modules from the source and target projects indistinguishable. Building upon this, DAM2P introduces a cross-domain kernel classifier based on the Maximum Margin Principle, which learns a hyperplane to distinguish between vulnerable

² <https://nvd.nist.gov/>

³ <https://www.cve.org/>

and non-vulnerable modules while minimizing the margin between the source and target projects. Since DAM2P and Dual-GD-DDAN investigate the same scenario, DAM2P maintains the same dataset setting as Dual-GD-DDAN.

LineVul [6]: Fu and Tantithamthavorn proposed LineVul, a Transformer-based fine-grained vulnerability detection method. A recent empirical study [55] shows that this method performs excellently. Therefore, we choose this method as a baseline. Since this method is designed for vulnerability detection in the within-project scenario, we apply it to the cross-project scenario while maintaining the same data settings as Dual-GD-DDAN.

ASSBert [49]: Sun et al. proposed ASSBert, a method that utilizes BERT to extract semantic representations of smart contract. This method integrates active learning and semi-supervised learning to build models for predicting vulnerabilities in smart contracts. We choose this method as a baseline because it uses BERT to extract semantic metrics and employs active learning approach, which is highly relevant to our research. It is worth noting that this method is designed for smart contract vulnerability detection in the within-project scenario. Therefore, we apply it to the cross-project scenario. Since this method also uses active learning to select samples, we keep the same data settings as CSVD-TF for a fair comparison.

VulBERT [52]: The VulBERT method detects vulnerabilities in source code by fine-tuning a pre-trained BERT model. It obtains semantic representations of the source code using the BERT model and performs prediction using a classifier. Since VulBERT is a within-project model, we follow the same setup as LineVul.

VulBERTa [53]: Hanif et al. proposed VulBERTa, a method for detecting security vulnerabilities in source code by fine-tuning an pre-trained RoBERTa model. The approach incorporates a customized tokenization pipeline and a masked language modeling task to capture the syntactic and semantic representations of C/C++ code, achieving strong performance on vulnerability detection tasks. Two types of classifiers were explored in their work, namely a Multi-Layer Perceptron (**VulBERTa-MLP**) and a TextCNN-based classifier (**VulBERTa-CNN**). In our study, both variants are adopted as baselines. Following the setup of LineVul, we apply these methods in a cross-project scenario and maintain the same data configuration as used in Dual-GD-DDAN.

For CSVD-TF, Dual-GD-DDAN, DAM2P, LineVul, VulBERT, VulBERTa-MLP and VulBERTa-CNN, we reproduce these methods based on their publicly available source codes. The original default parameters are adopted to ensure consistent and fair performance comparison. For ASSBert, since the original publication does not provide the source code, we manually implement the method according to the described procedures and available details. As the original paper does not specify the parameter settings, we optimize the parameters through a systematic grid search.

In summary, We consider a total of eight baselines. Among them, CSVD-TF, Dual-GD-DDAN, and DAM2P are CSVD methods, while ASSBert, LineVul, VulBERT, VulBERTa-MLP and VulBERTa-CNN are within-project methods. CSVD-TF is the first method proposed to select a portion of modules from the target project for labeling. Our baseline methods cover two scenarios in CSVD. Additionally, we apply ASSBert to CSVD, maintaining the same data settings as CSVD-TF. Therefore, our baseline methods are representative.

4.4. Performance measure

Our study uses the same evaluation metric as Cai et al. [20], which is the Area Under the receiver operator characteristic Curve (AUC). Previous research [56] has shown that in some cases, using precision, recall, and F1 score cannot accurately assess the performance of the model. This type of performance metric depends on the threshold used to determine the classification results, and the value of the metric can vary significantly when the threshold changes. In addition, when there is a class imbalance issue in the training set, these metrics

cannot accurately reflect the model's performance. Based on the above considerations, previous studies [20,56] used the AUC metric to assess the model's performance. The AUC metric does not rely on thresholds. It is computed by determining the area beneath the curve that plots the True Positive Rate (TPR) versus the False Positive Rate (FPR). The AUC score spans from 0 to 1. A larger AUC value implies superior model performance. An AUC of 1 signifies that the model can flawlessly differentiate between positive and negative samples. When the AUC is 0.5, it shows that the model is ineffective and performs comparably to random guessing. If the AUC is below 0.5, it means the model performs worse than random guessing and requires further modifications.

4.5. Experimental settings

To demonstrate the effectiveness of our proposed CSVD-AES in the field of CSVD, we select one project from all four projects as the source project (the labeled dataset used to train the model) and one project as the target project (the current project, assumed to have an unlabeled dataset). For example, the source project can be set as LibPNG, and the target project can be set as FFmpeg, VLC, or LibTIFF. We can obtain 12 different CSVD project combinations from the four projects. We denote each combination as “source project” → “target project” (e.g., FFmpeg → LibPNG, FFmpeg → VLC, etc.).

During the training process, we use stratified sampling, splitting 80% of the total dataset from each project as the training set and the remaining 20% as the test set. Unlike traditional CSVD methods, our experiments adopt a semi-cross-project setup, in which a small portion of labeled modules from the target project is used during model training. Under this setup, we use the training set from the source project as the initial labeled dataset, and the training set from the target project as the sample pool. We apply active learning methods to select data from the target project's training set and query labels, adding them to the labeled dataset for training. After the model is trained, we use the test set from the target project to evaluate the model. Consistent with previous study [20] on the selection ratio of target project data, we select 30% of the samples from the target project training set for labeling. In the experiments, we run the corresponding models for CSVD-AES and the baselines 5 times and report the averaged results.

We use the AdamW optimizer [57] with an initial learning rate set to 2e-5, and the mini-batch size is set to 16. The model is trained for a total of 10 epochs. All experiments are performed on a machine with an i7-12700K CPU and a GeForce RTX 4090D GPU, running Windows 10 as the operating system.

5. Experimental results

5.1. RQ1: How does our proposed CSVD-AES perform compared to the baselines?

Approach: In RQ1, we utilize the AUC to assess the performance of CSVD-AES and the eight baselines. For the seven baselines Dual-GD-DDAN [17], DAM2P [18], LineVul [6], ASSBert [49], VulBERTa-MLP [53], VulBERTa-CNN [53] and VulBERT [52]. We focus on the semantic metrics. This is in line with the settings used in their original studies. These methods do not incorporate expert metrics into their frameworks. For the CSVD-TF [20], we follow the original study's settings while considering both expert and semantic metrics. Since Dual-GD-DDAN, DAM2P, LineVul, VulBERTa-MLP, VulBERTa-CNN and VulBERT do not include a sample selection and labeling step, for a fair comparison, we use the same samples as CSVD-AES for training (For specific data settings, refer to Section 4.3). Since both CSVD-TF and ASSBert include a sample selection and labeling step, comparing with these two methods can validate the effectiveness of the sample selection in CSVD-AES. During the experiments, when the sample selection and labeling step is involved, we use the source project's training set as the initial training set and select samples from the target project's training

Table 3

The comparison results between CSVD-AES and eight baselines in terms of AUC.

Source→Target	CSVD-AES	CSVD-TF	LineVul	Dual-GD-DDAN	DAM2P	ASSBert	VulBERT	VulBERTa-MLP	VulBERTa-CNN
FFmpeg→LibPNG	0.978	0.918	0.966	0.873	0.916	0.992	0.897	0.907	0.935
FFmpeg→LibTIFF	0.820	0.711	0.700	0.679	0.749	0.712	0.773	0.770	0.762
FFmpeg→VLC	0.706	0.633	0.599	0.669	0.687	0.575	0.650	0.587	0.574
LibPNG→FFmpeg	0.810	0.758	0.623	0.761	0.773	0.685	0.731	0.779	0.785
LibPNG→LibTIFF	0.751	0.683	0.550	0.720	0.737	0.616	0.714	0.719	0.713
LibPNG→VLC	0.733	0.603	0.575	0.712	0.719	0.600	0.625	0.624	0.600
LibTIFF→FFmpeg	0.794	0.753	0.637	0.771	0.789	0.655	0.708	0.727	0.710
LibTIFF→LibPNG	0.949	0.835	0.895	0.912	0.891	0.910	0.901	0.972	0.932
LibTIFF→VLC	0.720	0.665	0.537	0.714	0.716	0.538	0.612	0.637	0.600
VLC→FFmpeg	0.834	0.755	0.507	0.500	0.782	0.755	0.509	0.509	0.502
VLC→LibPNG	0.876	0.813	0.631	0.500	0.861	0.995	0.589	0.771	0.784
VLC→LibTIFF	0.760	0.707	0.600	0.500	0.733	0.682	0.629	0.618	0.606
Average	0.811	0.740	0.652	0.693	0.780	0.726	0.695	0.718	0.708
p-value	—	***	***	***	***	***	***	***	***

Notes: *** means p -value < 0.001, ** means p -value < 0.01, * means p -value < 0.05.

set for labeling. We run the corresponding model 5 times for each method and report the average results. To ensure a fair comparison, we use the target project's test set to evaluate the performance of all models.

Results: Table 3 shows the comparison results between CSVD-AES and eight baselines. We find that the average AUC of CSVD-AES across all 12 CSVD project combinations is 0.811. Compared to CSVD-TF, LineVul, DAM2P, Dual-GD-DDAN, ASSBert, Vul-BERT, VulBERTa-MLP and VulBERTa-CNN, CSVD-AES improves performance by 9.6%, 24.4%, 4.0%, 17.0%, 11.7%, 16.7%, 13.0% and 14.5% respectively. Specifically, for each CSVD project combination, CSVD-AES outperforms the baselines in most cases, except for FFmpeg → LibPNG, LibTIFF → LibPNG, and VLC → LibPNG. It is noteworthy that in the CSVD project combinations FFmpeg → LibPNG and VLC → LibPNG, ASSBert achieves the best performance. ASSBert also uses active learning, which further confirms the effectiveness of active learning in the CSVD scenario. Additionally, we conducted the Wilcoxon signed-rank test [58] to further examine the statistical significance of the performance differences between CSVD-AES and five baselines, with a confidence level set at 95%. The Wilcoxon signed-rank test is used to compare whether the differences in means between two paired samples are statistically significant. If the computed p -value is less than 0.05, it indicates a significant difference between the two groups. Conversely, it suggests no significant difference between the samples. We used the Wilcoxon method from `scipy.stats` to calculate the p -values. In our study, we found that the computed p -values were all less than 0.001, indicating that CSVD-AES significantly outperforms the five baselines.

To more intuitively demonstrate the effectiveness of CSVD-AES, we use the Scott-Knott test [59] to group all methods based on their performance values, ensuring that the differences within each group are not significant (with a significance level set at 0.05). Specifically, it uses hierarchical clustering analysis to partition the methods into different groups. The higher the group and ranking, the better the method. Fig. 2 shows the results of the Scott-Knott test, with dotted lines representing the groups divided by the Scott-Knott test. The distribution of the AUC metric for the nine methods is displayed using the boxplot.

In Fig. 2, we observe that CSVD-AES and DAM2P are grouped together, and the methods in this group significantly outperform those in other groups. Moreover, CSVD-AES ranks ahead of DAM2P, indicating that CSVD-AES achieves better results.

Summary of RQ1

CSVD-AES can significantly outperform the five baselines, achieving performance improvements of 4.0% to 24.4% in terms of AUC. This indicates that using the CSVD-AES method in the CSVD scenario is effective.

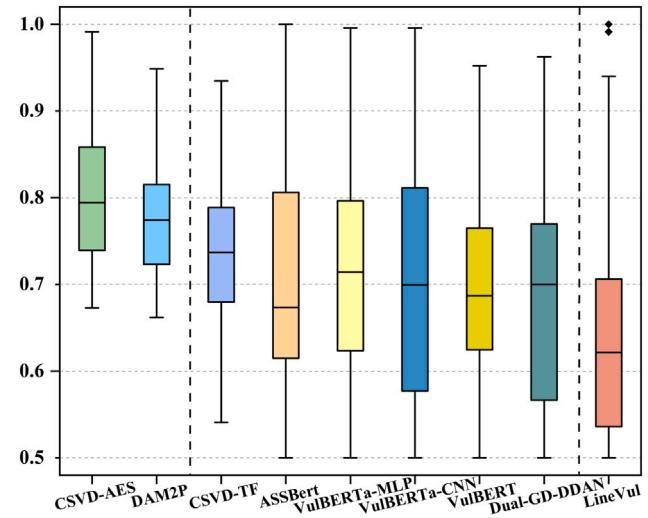


Fig. 2. The result of Scott-Knott test in terms of AUC.

5.2. RQ2: What impact does the incorporation of active learning have on the performance of CSVD-AES?

We conduct empirical studies from three aspects to answer RQ2. First, we want to compare the performance difference between using and not using active learning in the CSVD-AES. Second, we want to examine the effect of using different active learning methods on the performance of CSVD-AES to verify whether the active learning method we used has advantages. Third, we want to explore the impact of different proportion of samples selected from the target project using active learning on the performance of CSVD-AES.

5.2.1. RQ2.1: Whether using active learning can further boost the performance of CSVD-AES?

Approach: In this sub-RQ, we aim to investigate whether the use of active learning can improve the performance of the CSVD-AES. We remove the active learning phase from CSVD-AES while keeping the other processes unchanged. We refer to this method as CSVD-AES_{noal}. Additionally, the active learning method we introduce considers the class imbalance issue during the sampling process. To explore the effectiveness of the sampling balancing strategy, we remove this strategy from the active learning phase in CSVD-AES. The other processes remain unchanged. We refer to this method as CSVD-AES_{nobal}. We compare CSVD-AES with CSVD-AES_{noal} and CSVD-AES_{nobal}. Since we aim to investigate the impact of the sampling balancing strategy on CSVD-AES

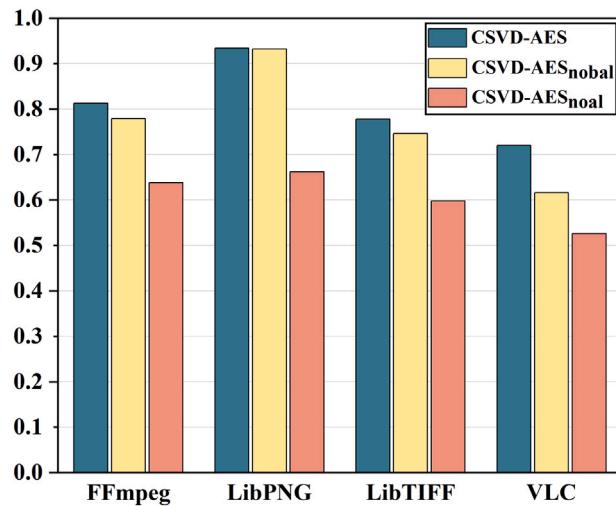


Fig. 3. Bar chart of CSVD-AES, CSVD-AES_{nobal}, and CSVD-AES_{noal} in terms of AUC performance.

Table 4

The comparison results between CSVD-AES, CSVD-AES_{nobal} and CSVD-AES_{noal} in terms of AUC.

Target project	CSVD-AES	CSVD-AES _{nobal}	CSVD-AES _{noal}
FFmpeg	0.813	0.779	0.638
LibPNG	0.934	0.932	0.662
LibTIFF	0.778	0.746	0.598
VLC	0.720	0.616	0.526
Average	0.811	0.768	0.606
p-value	—	***	***

Notes: *** means p -value < 0.001 , ** means p -value < 0.01 , * means p -value < 0.05 .

performance, we combine the results of 12 CSVD combinations with the same target project. We want to observe how the sampling balancing strategy affects the performance on target projects with different class imbalance ratios.

Results: Fig. 3 shows the performance comparison between CSVD-AES and two comparison methods across four target projects. From Fig. 3, we can see that both CSVD-AES and CSVD-AES_{nobal} perform better than CSVD-AES_{noal}. The performance of CSVD-AES and CSVD-AES_{nobal} is similar when the target project is LibPNG. However, when the target projects are FFmpeg, LibPNG, and VLC, CSVD-AES outperforms CSVD-AES_{nobal}. To provide a detailed comparison of the strengths and weaknesses of CSVD-AES and the comparison methods, we present the AUC values of all methods in Table 4.

Overall, the performance of CSVD-AES is better than both CSVD-AES_{nobal} and CSVD-AES_{noal}, with improvements of 5.6% and 33.8%, respectively. We also performed a Wilcoxon signed-rank test on the results. The results of the Wilcoxon signed-rank test indicate that these performance improvements are statistically significant. For the four target projects, we find that both CSVD-AES and CSVD-AES_{nobal} show significant performance improvements over CSVD-AES_{noal} (e.g., when the target project is fixed as FFmpeg, CSVD-AES improved by 27.4% compared to CSVD-AES_{noal}, and CSVD-AES_{nobal} improved by 22.1% compared to CSVD-AES_{noal}). This suggests that using Active Learning in CSVD-AES can significantly enhance the model's performance.

Compared to CSVD-AES_{nobal}, CSVD-AES improved by 4.4%, 0.2%, 4.3%, and 16.9% on the FFmpeg, LibPNG, LibTIFF, and VLC projects, respectively. We find that on the LibPNG dataset, the performance improvement of CSVD-AES was very small. This is because LibPNG has a small number of samples (only 495 in the training set) and a high number of vulnerable samples, resulting in a low class imbalance ratio. On the VLC dataset, we observed a 16.9% performance improvement

with CSVD-AES. Among the four projects, VLC has the largest class imbalance ratio, with vulnerable functions accounting for less than 1% of the total functions (as shown in Table 2). This suggests that the proposed sampling balancing strategy works better when the class imbalance issue is more pronounced. The above results indicate that applying the sampling balancing strategy during active learning can help mitigate the class imbalance problem in the target project.

Summary of RQ2.1

The empirical results demonstrate that the performance achieved using active learning methods outperforms that of methods without active learning. The sampling balancing strategy we propose proves to be effective, with its impact being particularly significant when the class imbalance issue in the target project is more pronounced.

5.2.2. RQ2.2: Is the uncertainty sampling used in CSVD-AES superior compared to other sampling strategies?

Approach: In this RQ, we aim to investigate whether the uncertainty sampling strategy we use outperforms other sampling strategies in active learning. We consider five different active learning sampling strategies: Random Sampling, Least Confidence [60], Margin Sampling [47], Entropy Sampling [48], and Wasserstein Adversarial Active Learning [50].

Random Sampling. In this sampling strategy, we randomly select code samples from the target project for labeling. The number of selected samples is the same as the uncertainty sampling strategy in CSVD-AES. We refer to the CSVD-AES using the Random Sampling strategy as CSVD-AES_{RS}.

Least Confidence [60]. In this sampling strategy, we select the code samples from the target project with the lowest model prediction confidence for labeling. The number of selected samples is the same as the uncertainty sampling strategy in CSVD-AES. We refer to the CSVD-AES using the Least Confidence strategy as CSVD-AES_{LC}.

Margin Sampling [47]. In this sampling strategy, we select the code samples from the target project that are closest to the model's decision boundary for labeling. The number of selected samples is the same as the uncertainty sampling strategy in CSVD-AES. We refer to the CSVD-AES using the Margin Sampling strategy as CSVD-AES_{MS}.

Entropy Sampling [48]. In this sampling strategy, we select the samples from the target project with the highest entropy in the model's predicted probability distribution for labeling. The number of selected samples is the same as the uncertainty sampling strategy in CSVD-AES. We refer to the CSVD-AES using the Entropy Sampling strategy as CSVD-AES_{ES}.

Wasserstein Adversarial Active Learning [50]. Shui et al. propose a deep active learning method called Wasserstein Adversarial Active Learning (WAAL). WAAL leverages unlabeled data information, constructs feature representations through adversarial training, and explicitly balances sample uncertainty and diversity during the query phase to select more informative samples for labeling. We use this method to select the same number of samples from the target project as the uncertainty sampling strategy in CSVD-AES for labeling. We refer to the CSVD-AES using the WAAL strategy as CSVD-AES_{WAAL}.

When comparing with these active learning sampling strategies, we keep the settings of other components unchanged and only modify the active learning sampling strategy to ensure a fair performance comparison.

Results: Fig. 4 uses a box plot to show the performance distribution of CSVD-AES using different active learning sampling strategies. In this figure, we use ▲ to indicate the average AUC value for each method. We can observe that CSVD-AES achieves the best performance compared to the other methods. Since the interquartile range of the box plot is smaller, indicating greater stability of CSVD-AES. This suggests that

Table 5

The comparison results between CSVD-AES and CSVD-AES using different sampling strategies in terms of AUC.

Target project	CSVD-AES	CSVD-AES _{RS}	CSVD-AES _{LC}	CSVD-AES _{MS}	CSVD-AES _{ES}	CSVD-AES _{WAAL}
FFmpeg	0.813	0.806	0.774	0.793	0.802	0.793
LibPNG	0.934	0.864	0.780	0.919	0.926	0.860
LibTIFF	0.778	0.733	0.723	0.770	0.777	0.740
VLC	0.720	0.662	0.682	0.704	0.674	0.718
Average	0.811	0.766	0.740	0.797	0.795	0.778
p-value	—	***	***	*	*	***

Notes: *** means p -value < 0.001 , ** means p -value < 0.01 , * means p -value < 0.05 .

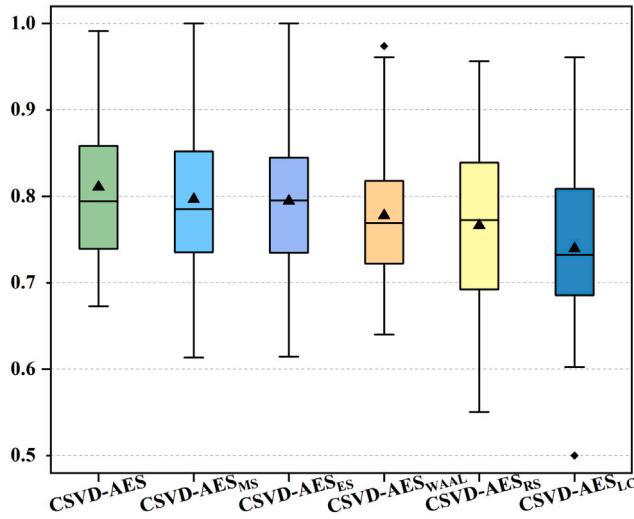


Fig. 4. The influence of using different active learning sampling strategies on the performance of CSVD-AES in terms of AUC.

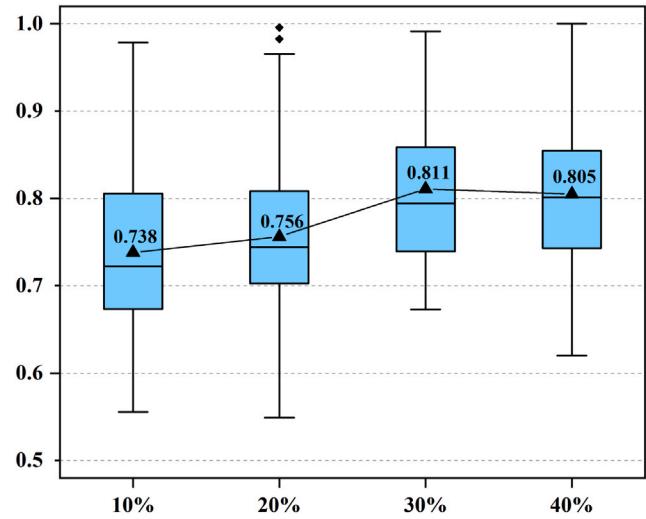


Fig. 5. The influence of using different target project sample selection ratios on the performance of CSVD-AES in terms of AUC.

the uncertainty sampling strategy we use outperforms the other active learning sampling strategies.

Table 5 shows the average AUC values of the CSVD-AES method for each target project using different sampling strategies. It is evident that for all target projects, CSVD-AES performs better with the current sampling strategy. It outperforms CSVD-AES with any other active learning sampling strategy. Overall, compared to CSVD-AES_{RS}, CSVD-AES_{LC}, CSVD-AES_{MS}, CSVD-AES_{ES}, and CSVD-AES_{WAAL}, CSVD-AES improves by 5.9%, 9.6%, 1.8%, 2.0%, and 4.2%, respectively. The results of the Wilcoxon signed-rank test also indicate that the performance improvement using the uncertainty sampling strategy is significantly better than that of using other active learning methods.

Summary of RQ2.2

The uncertainty sampling strategy we use improves performance by 1.8% to 9.6% compared to other active learning methods, further highlighting the effectiveness of the uncertainty sampling method.

5.2.3. RQ2.3: What impact does the proportion of sampled data from target project have on the performance of CSVD-AES?

Approach: In this RQ, we aim to explore whether the proportion of selected target project samples is reasonable. We select samples from the target project using the active learning method. The sample selection ranges from 10% to 40%, with a step size of 10%. We then compare their performance. To ensure a fair comparison, all settings are kept the same except for the sample selection proportion.

Results: **Fig. 5** presents a box plot showing the impact of different sample selection ratios from the target project on the performance of CSVD-AES. The horizontal axis represents the sample selection ratio,

Table 6

The influence of different sample selection ratios on the target project.

Target project	10%	20%	30%	40%
FFmpeg	0.753	0.783	0.813	0.810
LibPNG	0.838	0.855	0.934	0.917
LibTIFF	0.696	0.722	0.778	0.774
VLC	0.666	0.665	0.720	0.719

and the vertical axis represents the AUC value. We use ▲ to denote the average AUC value at each selection ratio. From the figure, it can be observed that the best performance is achieved when 30% of the samples are selected from the target project. When 40% of the samples are selected, the performance even decreases. The model performance improved by 2.4% from 10% to 20%, and by 7.3% from 20% to 30%. We further present the impact of different sample selection ratios on the target project in **Table 6**. We find that the best performance is achieved when the selection ratio is 30% for all target projects. Therefore, selecting 30% of the samples from the target project is a reasonable choice. Moreover, our sample selection ratio is consistent with previous study [20].

Summary of RQ2.3

Compared to selecting 10%, 20%, and 40% of the samples from the target project for labeling, selecting 30% of the samples for labeling achieved the best performance. These results suggest that a 30% selection ratio provides a balanced trade-off between data sufficiency and labeling cost, and is thus a reasonable configuration.

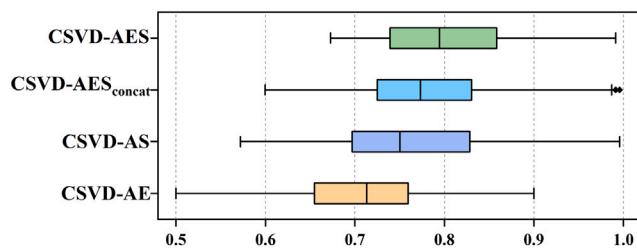


Fig. 6. The influence of fusing two types of metrics on the performance of CSVD-AES in terms of AUC.

Table 7

The comparison results between CSVD-AES, CSVD-AES_{concat}, CSVD-AE and CSVD-AS in terms of AUC.

Source→Target	CSVD-AES	CSVD-AES _{concat}	CSVD-AE	CSVD-AS
FFmpeg→LibPNG	0.978	0.971	0.826	0.968
FFmpeg→LibTIFF	0.820	0.795	0.664	0.746
FFmpeg→VLC	0.706	0.705	0.679	0.645
LibPNG→FFmpeg	0.810	0.787	0.780	0.758
LibPNG→LibTIFF	0.751	0.695	0.689	0.689
LibPNG→VLC	0.733	0.691	0.684	0.658
LibTIFF→FFmpeg	0.794	0.763	0.759	0.762
LibTIFF→LibPNG	0.949	0.931	0.789	0.928
LibTIFF→VLC	0.720	0.699	0.643	0.681
VLC→FFmpeg	0.834	0.825	0.712	0.775
VLC→LibPNG	0.876	0.862	0.528	0.865
VLC→LibTIFF	0.760	0.751	0.711	0.702
Average	0.811	0.789	0.705	0.765
p-value	–	**	***	***

Notes: *** means p -value < 0.001 , ** means p -value < 0.01 , * means p -value < 0.05 .

5.3. RQ3: Can the performance of CSVD-AES be improved by fusing two types of metrics?

Approach: In RQ3, we evaluate whether fusing the two types of metrics is effective. Specifically, we consider the cases where only expert metrics or only semantic metrics are used in the CSVD-AES. We refer to the CSVD-AES using only expert features as CSVD-AE, and the CSVD-AES using only semantic features as CSVD-AS. Additionally, we consider the case in CSVD-AES where the self-attention mechanism is not used, and the two types of features are simply concatenated. We refer to this as CSVD-AES_{concat}. In the comparison, except for differences in the code representation phase, the other module settings remain unchanged.

Results: Fig. 6 shows the performance distribution of CSVD-AES and comparison methods using boxplots. As we can see, compared to CSVD-AS and CSVD-AE which only use a single metric, CSVD-AES and CSVD-AES_{concat} achieve better results. This also demonstrates that fusing two metrics is more effective than using just one. We further present the average values of CSVD-AES and comparison methods across 12 CSVD combinations and perform the Wilcoxon signed-rank test. The experimental results are shown in Table 7.

From the table, we can see that compared to CSVD-AES_{concat}, CSVD-AE and CSVD-AS, CSVD-AES improves by 2.8%, 15.0% and 6.0%, respectively. The results of the Wilcoxon signed-rank test indicate that the performance improvement of CSVD-AES over the comparison methods is statistically significant. Additionally, CSVD-AES_{concat} shows improvements of 11.9% and 3.1% over CSVD-AE and CSVD-AS respectively. This further supporting the effectiveness of combining the two types of metrics. Across the 12 CSVD combinations, the performance of CSVD-AES also outperforms the comparison methods.

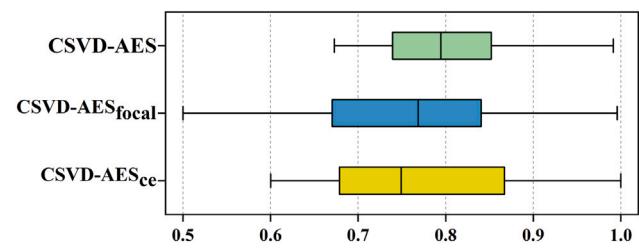


Fig. 7. The influence of using WCE loss function on the performance of CSVD-AES in terms of AUC.

Summary of RQ3

By using the self-attention mechanism to fuse expert metrics and semantic metrics, CSVD-AES can significantly improve performance in the CSVD scenario.

5.4. RQ4: How does the WCE loss function influence the performance of CSVD-AES?

Approach: In RQ4, we evaluate the contribution of the WCE loss function in CSVD-AES. We compare CSVD-AES with CSVD-AES using the cross-entropy loss function. We denote the CSVD-AES using the cross-entropy loss as CSVD-AES_{ce}. Additionally, we consider the focal loss, which has been used in the research of Chen et al. [54]. We compare CSVD-AES with CSVD-AES using the focal loss function, and denote it as CSVD-AES_{focal}. To ensure a fair comparison, all components except for the loss function are kept the same.

Results: Fig. 7 shows a boxplot comparing CSVD-AES, CSVD-AES_{focal}, and CSVD-AES_{ce}. As can be seen, CSVD-AES outperforms both comparison methods and demonstrates more stable performance. We further present the performance of CSVD-AES and the comparison methods in a table, as shown in Table 8.

From this table, we can see that CSVD-AES achieves the best performance. Compared to CSVD-AES_{focal} and CSVD-AES_{ce}, the performance of CSVD-AES improved by 7.0% and 5.2%, respectively. The results of the Wilcoxon signed-rank test indicate that CSVD-AES significantly outperforms CSVD-AES_{focal} and CSVD-AES_{ce}. Considering the average performance across the 12 CSVD combinations, CSVD-AES outperforms the comparison methods in most cases. Although Focal loss is a loss function designed to handle class imbalance, it did not achieve satisfactory results in the context of this study. Combining Fig. 7 and Table 8, we find that the performance with Focal loss is quite unstable. The boxplot of CSVD-AES_{focal} has a longer box. In Table 8, only a few combinations (e.g. FFmpeg→LibTIFF, LibTIFF→VLC) perform better than CSVD-AES_{ce}. We further calculate the variance of CSVD-AES and CSVD-AES_{focal}, and find that the variance of CSVD-AES is 0.0078, while the variance of CSVD-AES_{focal} is 0.0138. Therefore, using WCE loss function results in better and more stable performance compared to focal loss function.

Summary of RQ4

The use of the WCE loss function in CSVD-AES can effectively alleviate the class imbalance problem in CSVD. Compared to cross-entropy loss function and focal loss function, using the WCE loss function in CSVD-AES achieves better and more stable performance.

Table 8

The comparison results between CSVD-AES, CSVD-AES_{focal} and CSVD-AES_{ce} in terms of AUC.

Source→Target	CSVD-AES	CSVD-AES _{focal}	CSVD-AES _{ce}
FFmpeg→LibPNG	0.978	0.973	0.975
FFmpeg→LibTIFF	0.820	0.765	0.763
FFmpeg→VLC	0.706	0.622	0.670
LibPNG→FFmpeg	0.810	0.765	0.742
LibPNG→LibTIFF	0.751	0.708	0.710
LibPNG→VLC	0.733	0.642	0.642
LibTIFF→FFmpeg	0.794	0.760	0.724
LibTIFF→LibPNG	0.949	0.874	0.955
LibTIFF→VLC	0.720	0.709	0.630
VLC→FFmpeg	0.834	0.859	0.860
VLC→LibPNG	0.876	0.777	0.848
VLC→LibTIFF	0.760	0.642	0.737
Average	0.811	0.758	0.771
p-value	—	***	***

Notes: *** means p -value < 0.001, ** means p -value < 0.01, * means p -value < 0.05.

Table 9

Statistical information of Chrome and Qemu datasets.

Project	# Vulnerable functions	# Non-vulnerable functions	% Vulnerable functions
Chrome	2965	63 265	4.477%
Qemu	5672	7647	42.586%

6. Discussion

6.1. Generalization of CSVD-AES

In this subsection, we evaluate the generalization capability of CSVD-AES on large-scale datasets. To this end, we conduct experiments on the Chrome and Qemu datasets. The Chrome dataset is derived from Big-Vul [61], a comprehensive dataset constructed from over 300 open-source C/C++ projects on GitHub. We choose Chrome due to its relatively large data volume. The Qemu dataset is obtained from Devign [62], which consists of manually labeled functions extracted from four large-scale open-source C projects. We select Qemu as a representative target for our analysis. The statistical characteristics of the Chrome and Qemu datasets are summarized in Table 9.

We re-evaluate CSVD-AES and eight baselines on the Chrome and Qemu datasets, maintaining the same experimental settings as in RQ1. To ensure a fair comparison, we fix the random seed across all nine methods and adopt AUC as the evaluation metric. The results are summarized in Table 10. In the Chrome → Qemu setting, CSVD-AES achieves AUC improvements of 36.2%, 10.0%, 27.1%, 26.6%, 12.7%, 3.7%, 3.7%, and 3.3% over CSVD-TF, LineVul, Dual-GD-DDAN, DAM2P, ASSBert, VulBERT, VulBERTa-MLP, and VulBERTa-CNN, respectively. In the Qemu → Chrome setting, the corresponding improvements are 41.7%, 34.2%, 33.7%, 35.4%, 8.6%, 2.0%, 0.3%, and 8.3%. Overall, BERT-based approaches (e.g., CSVD-AES, ASSBert, and VulBERT) consistently outperform conventional deep learning methods (e.g., Dual-GD-DDAN and DAM2P) on large-scale datasets. Notably, even with the same training data, CSVD-AES surpasses other BERT-based methods, highlighting its superior effectiveness.

6.2. Comparison with within-project vulnerability detection

In this subsection, we investigate the performance of CSVD-AES in the within-project vulnerability detection setting. To facilitate comparison with Table 3 in RQ1, we report the AUC scores of CSVD-AES and all baselines under the within-project setting. Since CSVD-AES, CSVD-TF, and ASSBert involve selecting program modules from the target project, we remove the module selection step in this setting. To ensure

Table 10

The comparison results between CSVD-AES and eight baselines on the Chrome and Qemu datasets.

Method	Chrome → Qemu	Qemu → Chrome
CSVD-AES	0.681	0.781
CSVD-TF	0.500	0.551
LineVul	0.619	0.582
Dual-GD-DDAN	0.536	0.584
DAM2P	0.538	0.577
ASSBert	0.604	0.719
VulBERT	0.657	0.766
VulBERTa-MLP	0.657	0.779
VulBERTa-CNN	0.659	0.721

the robustness of our results, each experiment is repeated five times, and the average performance is reported. The AUC results of CSVD-AES and all baselines in the within-project scenario are shown in Table 11.

As shown in Table 11, under the within-project setting, CSVD-AES outperforms all baselines on the FFmpeg, LibTIFF, and VLC projects. The advantage of CSVD-AES is particularly evident on the VLC project. Due to the severe class imbalance in VLC, most models perform poorly, highlighting the effectiveness of the WCE loss function. On the LibPNG dataset, CSVD-AES performs slightly worse than models such as Dual-GD-DDAN. This is likely because LibPNG contains a small amount of data, allowing most models to achieve relatively good results. Overall, CSVD-AES achieves performance improvements ranging from 3.0% to 16.5% compared to all baselines. By comparing the results in Table 11 with those in Table 3, we observe that even though a portion of the target project's modules is added to the source project during training under the cross-project setting, the within-project performance consistently surpasses the cross-project performance. This further demonstrates the inherent challenge of CSVD.

In addition, as shown in Table 5, under the cross-project setting with a fixed target project, CSVD-AES achieves the AUC of 0.813 on FFmpeg, 0.778 on LibTIFF, and 0.720 on VLC. These results even outperform those of many baselines under the within-project setting.

6.3. Evaluation on F1 performance measure

In this subsection, we aim to investigate the performance of CSVD-AES in terms of the F1 score. The F1 score provides a comprehensive evaluation of a model's Precision and Recall by computing their harmonic mean. A higher F1 score indicates better overall performance in both the accuracy and completeness of vulnerability detection. The F1 score is calculated as follows:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (10)$$

where Precision and Recall are calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \quad (11)$$

Here, TP, FP, and FN denote the number of true positives, false positives, and false negatives, respectively.

We adopt the same experimental setup as described in RQ1 for evaluating CSVD-AES and the eight baselines, with the only modification being the use of F1 score as the evaluation metric. To more intuitively illustrate the performance differences among the methods, we employ the Scott-Knott test to statistically group the approaches based on their F1 scores. Fig. 8 presents the results, where the dotted lines indicate the groupings derived from the Scott-Knott test. Overall, BERT-based methods demonstrate superior performance compared other approaches, and are grouped into the first and second tiers. Among them, LineVul shows slightly lower performance relative to other BERT-based methods. Notably, CSVD-AES ranks in the top group and consistently outperforms its BERT-based counterparts. This improvement is attributed to the integration of expert and semantic

Table 11

The comparison results between CSVD-AES and eight baselines in the within-project setting.

Project	CSVD-AES	CSVD-TF	LineVul	Dual-GD-DDAN	DAM2P	ASSBert	VulBERT	VulBERTa-MLP	VulBERTa-CNN
FFmpeg	0.856	0.778	0.793	0.806	0.843	0.779	0.763	0.751	0.756
LibPNG	0.999	0.978	0.999	1.000	1.000	1.000	0.998	1.000	1.000
LibTIFF	0.807	0.735	0.756	0.786	0.800	0.754	0.743	0.781	0.741
VLC	0.916	0.666	0.625	0.500	0.832	0.625	0.662	0.661	0.575
Average	0.895	0.789	0.793	0.773	0.869	0.789	0.791	0.798	0.768

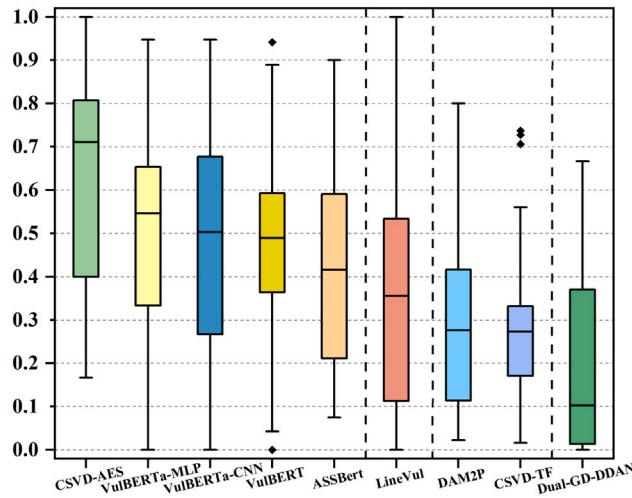


Fig. 8. The result of Scott-Knott test in terms of F1.

metrics, as well as the use of the WCE loss function. Specifically, CSVD-AES achieves F1 scores that are 3.9% to 179% higher than those of the baseline models. The observed differences between groups are statistically significant, validating the effectiveness of CSVD-AES in enhancing vulnerability detection performance.

6.4. Effectiveness of sampling balancing strategy

In this subsection, we evaluate the effectiveness of the proposed sampling balancing strategy, which is employed during the active learning phase. In the context of active learning, the presence of class imbalance in the target project poses a challenge: selecting program modules from the target project for labeling and integrating them into the source project may further exacerbate the imbalance among labeled modules. To address this issue, we introduce a sampling balancing strategy aimed at mitigating the impact of such skewed distributions. However, a systematic comparison with conventional techniques, such as random undersampling [63] and random oversampling [63], has not yet been performed. To this end, we denote the CSVD-AES variants utilizing random undersampling and random oversampling as CSVD-AES_{us} and CSVD-AES_{os}, respectively. Following the experimental setup of RQ2.1, we aggregate the results from 12 CSVD combinations sharing the same target project. The detailed experimental results are presented in Table 12.

In Table 12, we also consider the CSVD-AES_{nobal} variant introduced in RQ2.1, which does not employ any class imbalance mitigation strategy on the selected program modules. The results show that CSVD-AES consistently outperforms the other approaches on the FFmpeg, LibTIFF, and VLC datasets. Although CSVD-AES_{os} achieves the best performance on the LibPNG dataset, its effectiveness on the FFmpeg and VLC datasets is inferior to that of CSVD-AES_{nobal}. Overall, CSVD-AES yields the best performance in terms of the AUC metric across the majority of datasets, and the observed improvements are statistically significant. These findings demonstrate the effectiveness of the proposed sampling balancing strategy in mitigating class imbalance during the active learning process.

Table 12

The comparison results between CSVD-AES, CSVD-AES_{nobal}, CSVD-AES_{us}, and CSVD-AES_{os} in terms of AUC.

Target project	CSVD-AES	CSVD-AES _{nobal}	CSVD-AES _{us}	CSVD-AES _{os}
FFmpeg	0.813	0.779	0.801	0.758
LibPNG	0.934	0.932	0.911	0.975
LibTIFF	0.778	0.746	0.750	0.766
VLC	0.720	0.616	0.700	0.608
Average	0.811	0.768	0.790	0.777
p-value	—	***	**	***

Notes: *** means p -value < 0.001 , ** means p -value < 0.01 , * means p -value < 0.05 .

6.5. Threats to validity

External validity: The primary external threat to this study lies in the reliability of labeled data, as label inaccuracies may affect classification performance. To address this, we conduct experiments on four real-world software projects that have been widely used in prior CSVD studies [17,18,20], ensuring the credibility of our evaluation. Additionally, the dataset suffers from class imbalance, which may hinder model training. Our approach tackles this challenge by incorporating both a loss function tailored for imbalance and an active learning strategy.

Internal validity: One internal threat involves the selection of target modules for labeling, as the effectiveness of CSVD-AES may be influenced by the uncertainty sampling strategy. To reduce this risk, we repeat each experiment five times with different random seeds and report the averaged results. Another threat relates to hyperparameter settings. We adopt default configurations for baselines and apply random grid search to optimize CSVD-AES. While exhaustive search is infeasible, the chosen settings consistently outperform the baselines, indicating reasonable robustness.

Conclusion validity: A potential threat to conclusion validity concerns the appropriateness of baseline selection. To address this, we include state-of-the-art CSVD methods, such as DualGD-DDAN [17] and DAM2P [18], as domain adaptation-based baselines. CD-VulD and CPVD are excluded due to unavailable data or code. We also incorporate CSVD-TF [20], which leverages metric fusion and shares a similar setting, and ASSBert [49], which employs active learning in a within-project context. Additionally, we include VulBERTa-MLP [53], LineVul [6], VulBERT, and VulBERTa-CNN [53] as baselines, given their strong performance in the within-project setting.

Another threat concerns the fairness of model comparisons. We divide baselines into those requiring target project data selection and those that do not. For the former, we use the same number of selected samples as CSVD-AES; for the latter, we train with both source data and the selected target samples. In ablation studies, only the components under comparison are modified, while others remain fixed to ensure fairness.

Construct validity: The primary construct threat arises from potential implementation flaws in CSVD-AES and the baselines. To mitigate this, we rely on open-source code for implementing baselines whenever available; however, the code for ASSBert was not accessible. Consequently, for both CSVD-AES and ASSBert, we utilize well-established libraries to implement each component and perform thorough code inspections to ensure correctness.

7. Related work

7.1. Cross-project software vulnerability detection

CSVD seeks to utilize program modules from the source project to train models capable of predicting vulnerable modules in a target project. Prior research has primarily employed software metrics or text mining techniques to construct machine learning models for CSVD [64]. Moshtari et al. [65] examined 15 software metrics encompassing unit complexity and coupling, and evaluated eight methods across both within-project and cross-project vulnerability detection scenarios. Their comparative analysis revealed that complexity metrics exhibit superior predictive power over coupling metrics in CSVD tasks. To further improve recall, Moshtari et al. [66] introduced the IVH metric and incorporated it into the complexity metric set. Walden et al. [67] contrasted text mining-based models with those relying on software metrics as predictive features, finding that the former consistently achieved higher recall rates. Additionally, Kalouptsoglou et al. [68] integrated software metrics with deep learning techniques, resulting in a CSVD model demonstrating enhanced performance.

Driven by the rapid progress in deep learning and transfer learning, recent studies have devoted significant efforts to mitigating the distributional discrepancies between source and target projects, thereby enhancing the generalization capability of cross-project vulnerability detection models. Liu et al. [15] proposed CD-VulD, which leverages a Bi-LSTM network to extract semantic metrics and applies metric transfer learning to reduce distributional discrepancies between projects. Nguyen et al. [16] proposed SCDAN, which combines deep domain adaptation with automated semantic metric learning for vulnerability detection. It also adopts semi-supervised learning to exploit unlabeled data in the target project. Nguyen et al. [17] further proposed Dual-GD-DDAN, which employs dual generators and discriminators to map source and target data into a shared feature space, effectively reducing distribution discrepancies while mitigating data degradation and boundary distortion. Nguyen et al. [18] proposed DAM2P, which leverages a GAN-based framework to align source and target code in a shared feature space, and introduces a cross-domain kernel classifier based on the Maximum Margin Principle to distinguish vulnerable modules while reducing domain discrepancy. Zhang et al. [19] proposed CPVD, which uses code property graphs and graph neural networks to learn code representations. It employs domain adaptation to mitigate distribution gaps between source and target projects. Cai et al. [20] proposed CSVD-TF, which enhances CSVD performance by fusing expert and semantic metrics. It also applies TrAdaBoost with a small labeled subset of the target project to reduce distributional differences.

Previous studies have explored fusing expert and semantic metrics and labeling partial target data, but often rely on manually tuned model weights and static semantic feature extraction, limiting adaptability. Moreover, random sampling may overlook informative target samples. To address these limitations, we propose a self-attention-based fusion [23] of expert and semantic features, fine-tune CodeBERT [39] on labeled SVD data to enhance semantic understanding, and apply active learning to select valuable target samples for labeling. To mitigate class imbalance in CSVD, we adopt a WCE loss to emphasize minority classes and introduce a sampling balancing strategy during active learning.

7.2. Active learning for software engineering

In recent years, active learning techniques have been increasingly adopted in the field of software engineering. To alleviate the time-consuming and labor-intensive process of literature screening during systematic literature reviews, Yu et al. [69] proposed the FASTREAD method, which integrates uncertainty sampling and certainty sampling to automatically identify relevant studies, thereby significantly reducing manual effort. They further introduced the HARMLESS [70], which employs active learning to prioritize code segments that are more

likely to contain vulnerabilities, thus minimizing the time and resources required for vulnerability detection. Hu et al. [71] introduced the concept of Active Code Learning, in which they surveyed 11 acquisition functions from existing active learning studies and adapted them for code-related tasks. Kang et al. [72] developed the ALP method to detect Java API misuses by representing programs as graphs and mining discriminative subgraphs, while leveraging active learning to reduce the labeling burden. Zhou et al. [73] designed a tool named BRAID that combines learning-to-rank and active learning techniques to enhance the performance of recommendation systems.

Although active learning techniques have been applied in the software engineering domain, to the best of our knowledge, no prior studies have investigated their application in the context of CSVD. In our work, we employ active learning to select a small subset of representative modules from the target project for expert annotation. By incorporating both the source project and the newly labeled target modules, we train the SVD model to improve its effectiveness in the CSVD scenario.

8. Conclusion and future work

In previous CSVD studies, researchers have attempted to label a small amount of data from the target project and add it to the source project to train the model and improve its performance. However, how to systematically select representative modules for labeling has not received sufficient attention in existing research. Our research proposes a novel method, CSVD-AES. It aims to strategically select a subset of modules from the target project for labeling using active learning. Additionally, it considers a more effective fusion of expert and semantic metrics. Specifically, in the code representation phase, we use a self-attention mechanism to fuse expert metrics and semantic metrics. In the model construction phase, to address the class imbalance problem, we employ WCE Loss function to make the model focus more on the minority class samples during training. Finally, in the active learning phase, we use uncertainty sampling strategy to sample and label from the target project. During the labeling process, we apply a sampling balancing strategy to ensure the balance of the modules obtained from the target project. The final experimental results demonstrate that CSVD-AES outperforms the five baseline methods we considered. A series of ablation experiments further validate the rationale behind the component settings in CSVD-AES.

In the future, we plan to expand our research in three ways. First, we aim to collect more practical software project data to verify the validity of CSVD-AES in cross-project software vulnerability detection for other programming languages (e.g., Python and Rust). We can also explore the issue of vulnerability detection across languages, as different languages often share similar structural logic. Second, we can attempt to integrate more code feature representations, such as considering the fusion of Program Dependence Graph (PDG). Finally, we may consider using federated learning [74,75] to address the privacy issues related to code data. In practice, different organizations often do not make their code data publicly available. How to share data across projects to train effective vulnerability detection models is an urgent problem that needs to be solved.

CRediT authorship contribution statement

Zhidan Yuan: Writing – original draft, Software, Data curation.
Xiang Chen: Writing – review & editing, Supervision, Methodology, Conceptualization.
Juan Zhang: Validation, Software, Data curation.
Weiming Zeng: Writing – review & editing, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Zidan Yuan reports financial support was provided by Natural Science Foundation of the Jiangsu Higher Education Institutions of China. Reports a relationship with that includes: Has patent pending to. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported in part by Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant No. 23KJD580001).

Data availability

Data will be made available on request.

References

- [1] B. Potter, G. McGraw, Software security testing, *IEEE Secur. Priv.* 2 (5) (2004) 81–85.
- [2] H. Gabriel, Analyza a demonstrace zranitelnosti proxylogen (B.S. thesis), České vysoké učenítechnické v Praze. Výpočetní informačnícentrum., 2022.
- [3] B. Steenhoek, M.M. Rahman, R. Jiles, W. Le, An empirical study of deep learning models for vulnerability detection, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, 2023, pp. 2237–2248.
- [4] Y. Li, S. Wang, T.N. Nguyen, Vulnerability detection with fine-grained interpretations, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, Association for Computing Machinery, New York, NY, USA, 2021, pp. 292–303.
- [5] N. Shiri Harzevili, A. Boaye Belle, J. Wang, S. Wang, Z.M.J. Jiang, N. Nagappan, A systematic literature review on automated software vulnerability detection using machine learning, *ACM Comput. Surv.* 57 (3) (2024).
- [6] M. Fu, C. Tantithamthavorn, LineVul: A transformer-based line-level vulnerability prediction, in: 2022 IEEE/ACM 19th International Conference on Mining Software Repositories, MSR, 2022, pp. 608–620, <http://dx.doi.org/10.1145/3524842.3524852>.
- [7] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, Y. Zhong, Vuldeepecker: A deep learning-based system for vulnerability detection, 2018, arXiv preprint [arXiv:1801.01681](https://arxiv.org/abs/1801.01681).
- [8] CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection, *J. Syst. Softw.* 199 (2023) 111623.
- [9] M. Fu, C.K. Tantithamthavorn, V. Nguyen, T. Le, ChatGPT for vulnerability detection, classification, and repair: How far are we? in: 2023 30th Asia-Pacific Software Engineering Conference, APSEC, 2023, pp. 632–636.
- [10] K. Tamberg, H. Bahsi, Harnessing large language models for software vulnerability detection: A comprehensive benchmarking study, 2024, [arXiv:2405.15614](https://arxiv.org/abs/2405.15614).
- [11] X. Yin, C. Ni, S. Wang, Multitask-based evaluation of open-source LLM on software vulnerability, *IEEE Trans. Softw. Eng.* 50 (11) (2024) 3071–3087.
- [12] R. Croft, M.A. Babar, M.M. Khloosi, Data quality for software vulnerability datasets, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, 2023, pp. 121–133.
- [13] X. Wang, R. Hu, C. Gao, X.-C. Wen, Y. Chen, Q. Liao, Reposvul: A repository-level high-quality vulnerability dataset, in: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, 2024, pp. 472–483.
- [14] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, Y. Chen, Vulnerability detection with code language models: How far are we?, 2024, arXiv preprint [arXiv:2403.18624](https://arxiv.org/abs/2403.18624).
- [15] S. Liu, G. Lin, L. Qu, J. Zhang, O. De Vel, P. Montague, Y. Xiang, CD-VulD: Cross-domain vulnerability discovery based on deep domain adaptation, *IEEE Trans. Dependable Secur. Comput.* 19 (1) (2020) 438–451.
- [16] V. Nguyen, T. Le, T. Le, K. Nguyen, O. DeVel, P. Montague, L. Qu, D. Phung, Deep domain adaptation for vulnerable code function identification, in: 2019 International Joint Conference on Neural Networks, IJCNN, IEEE, 2019, pp. 1–8.
- [17] V. Nguyen, T. Le, O. de Vel, P. Montague, J. Grundy, D. Phung, Dual-component deep domain adaptation: A new approach for cross project software vulnerability detection, in: Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part I 24, Springer, 2020, pp. 699–711.
- [18] V. Nguyen, T. Le, C. Tantithamthavorn, J. Grundy, D. Phung, Deep domain adaptation with max-margin principle for cross-project imbalanced software vulnerability detection, *ACM Trans. Softw. Eng. Methodol.* 33 (6) (2024).
- [19] C. Zhang, B. Liu, Y. Xin, L. Yao, CPVD: Cross project vulnerability detection based on graph attention network and domain adaptation, *IEEE Trans. Softw. Eng.* 49 (8) (2023) 4152–4168, <http://dx.doi.org/10.1109/TSE.2023.3285910>.
- [20] Z. Cai, Y. Cai, X. Chen, G. Lu, W. Pei, J. Zhao, CSVD-TF: Cross-project software vulnerability detection with TrAdaBoost by fusing expert metrics and semantic metrics, *J. Syst. Softw.* 213 (2024) 112038.
- [21] W. Dai, Q. Yang, G.-R. Xue, Y. Yu, Boosting for transfer learning, in: Proceedings of the 24th International Conference on Machine Learning, ICML '07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 193–200.
- [22] S.-J. Huang, R. Jin, Z.-H. Zhou, Active learning by querying informative and representative examples, *Neural Inf. Process. Syst.* (2010).
- [23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS '17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 6000–6010.
- [24] A. Desai, G. Gold, B. Hargreaves, A. Chaudhari, Technical considerations for semantic segmentation in MRI using convolutional neural networks, 2019, arXiv: Image and Video Processing.
- [25] G. Lin, J. Zhang, W. Luo, L. Pan, Y. Xiang, O. De Vel, P. Montague, Cross-project transfer representation learning for vulnerable function discovery, *IEEE Trans. Ind. Informatics* 14 (7) (2018) 3289–3297, <http://dx.doi.org/10.1109/TII.2018.2821768>.
- [26] Y. Ganin, V. Lempitsky, Unsupervised domain adaptation by backpropagation, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML '15, JMLR.org, 2015, pp. 1180–1189.
- [27] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, et al., Xgboost: extreme gradient boosting, *R Packag. Version 0.4-2 1 (4)* (2015) 1–4.
- [28] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, *J. Artif. Int. Res.* 16 (1) (2002) 321–357.
- [29] B. Turhan, A.T. Misirlı, A. Bener, Empirical evaluation of the effects of mixed project data on learning defect predictors, *Inf. Softw. Technol.* 55 (6) (2013) 1101–1118.
- [30] D. Ryu, J.-I. Jang, J. Baik, A transfer cost-sensitive boosting approach for cross-project defect prediction, *Softw. Qual. J.* 25 (1) (2017) 235–272.
- [31] X. Xia, D. Lo, S.J. Pan, N. Nagappan, X. Wang, Hydra: Massively compositional model for cross-project defect prediction, *IEEE Trans. Softw. Eng.* 42 (10) (2016) 977–998.
- [32] L. Chen, B. Fang, Z. Shang, Y. Tang, Negative samples reduction in cross-company software defects prediction, *Inf. Softw. Technol.* 62 (2015) 67–77.
- [33] A. Abbad-Andalousi, On the relationship between source-code metrics and cognitive load: A systematic tertiary review, *J. Syst. Softw.* 198 (2023) 111619.
- [34] L. Ardito, R. Coppola, L. Barbato, D. Verga, A tool-based perspective on software code maintainability metrics: A systematic literature review, *Sci. Program.* 2020 (1) (2020) 8840389.
- [35] A. Kaur, A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes, *Arch. Comput. Methods Eng.* 27 (4) (2020) 1267–1296.
- [36] A. Bexell, Software source code readability: A mapping study, 2020.
- [37] R.P. Buse, W.R. Weimer, A metric for software readability, in: Proceedings of the 2008 International Symposium on Software Testing and Analysis, 2008, pp. 121–130.
- [38] A.S. Nuñez-Varela, H.G. Pérez-Gonzalez, F.E. Martínez-Perez, C. Soubervielle-Montalvo, Source code metrics: A systematic mapping study, *J. Syst. Softw.* 128 (2017) 164–197.
- [39] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, CodeBERT: A pre-trained model for programming and natural languages, in: T. Cohn, Y. He, Y. Liu (Eds.), Findings of the Association for Computational Linguistics, EMNLP 2020, Association for Computational Linguistics, Online, 2020, pp. 1536–1547.
- [40] Z. Gao, X. Xia, D. Lo, J. Grundy, T. Zimmermann, Automating the removal of obsolete TODO comments, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021.
- [41] K. Liu, G. Yang, X. Chen, Y. Zhou, EL-CodeBert: Better exploiting CodeBert to support source code-related classification tasks, in: Proceedings of the 13th Asia-Pacific Symposium on Internetwork, 2022, pp. 147–155.
- [42] X. Yang, S. Wang, Y. Li, S. Wang, Does data sampling improve deep learning-based vulnerability detection? Yeas! and Nays!.
- [43] C. Yu, G. Yang, X. Chen, K. Liu, Y. Zhou, BashExplainer: Retrieval-augmented bash code comment generation based on fine-tuned CodeBERT, 2022.

- [44] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North, 2019.
- [45] C. Ni, W. Wang, K. Yang, X. Xia, K. Liu, D. Lo, The best of both worlds: integrating semantic features with expert features for defect prediction and localization, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Association for Computing Machinery, New York, NY, USA, 2022, pp. 672–683.
- [46] X. Wu, Z.-H. Cao, T.-Z. Huang, L.-J. Deng, J. Chanussot, G. Vivone, Fully-connected transformer for multi-source image fusion, 47 (3) (2025) 2071–2088.
- [47] T. Scheffer, C. Decomain, S. Wrobel, Active hidden Markov models for information extraction, in: Advances in Intelligent Data Analysis, Lecture Notes in Computer Science, 2001, pp. 309–318.
- [48] B. Settles, Active learning literature survey, 2009.
- [49] X. Sun, L. Tu, J. Zhang, J. Cai, B. Li, Y. Wang, ASSBERT: Active and semi-supervised bert for smart contract vulnerability detection, J. Inf. Secur. Appl. (2023) 103423.
- [50] C. Shui, F. Zhou, C. Gagné, B. Wang, Deep active learning: Unified and principled method for query and training, 2019, arXiv: Learning.
- [51] X.-C. Wen, X. Wang, C. Gao, S. Wang, Y. Liu, Z. Gu, When less is enough: Positive and unlabeled learning model for vulnerability detection, in: Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering, ASE '23, IEEE Press, 2024, pp. 345–357.
- [52] S. Shafiq, vulBERT, 2021, <https://github.com/SamanShafiq/vulBERT>.
- [53] H. Hanif, S. Maffeis, VulBERTa: Simplified source code pre-training for vulnerability detection, in: 2022 International Joint Conference on Neural Networks, IJCNN, 2022, pp. 1–8, <http://dx.doi.org/10.1109/IJCNN55064.2022.9892280>.
- [54] X. Chen, F. Xu, Y. Huang, N. Zhang, Z. Zheng, JIT-smart: A multi-task learning framework for just-in-time defect prediction and localization, Proc. ACM Softw. Eng. 1 (FSE) (2024).
- [55] B. Steenhoeck, M.M. Rahman, R. Jiles, W. Le, An empirical study of deep learning models for vulnerability detection, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, IEEE, 2023, pp. 2237–2248.
- [56] C. Tantithamthavorn, A.E. Hassan, K. Matsumoto, The impact of class rebalancing techniques on the performance and interpretation of defect prediction models, IEEE Trans. Softw. Eng. (2020) 1200–1219.
- [57] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, la, USA, May 6–9, 2019, OpenReview.net, 2019.
- [58] F. Wilcoxon, Individual comparisons by ranking methods, in: Breakthroughs in Statistics: Methodology and Distribution, Springer, 1992, pp. 196–202.
- [59] E. Jelihovschi, J.C. Faria, I.B. Allaman, ScottKnott: A package for performing the scott-knott clustering algorithm in r, TEMA (São Carlos) (2015) 003.
- [60] D.D. Lewis, W.A. Gale, A sequential algorithm for training text classifiers, in: SIGIR '94, 1994, pp. 3–12.
- [61] J. Fan, Y. Li, S. Wang, T.N. Nguyen, AC/C++ code vulnerability dataset with code changes and CVE summaries, in: Proceedings of the 17th International Conference on Mining Software Repositories, 2020, pp. 508–512.
- [62] Y. Zhou, S. Liu, J. Siow, X. Du, Y. Liu, Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks, Adv. Neural Inf. Process. Syst. 32 (2019).
- [63] R. Mohammed, J. Rawashdeh, M. Abdullah, Machine learning with oversampling and undersampling techniques: overview study and experimental results, in: 2020 11th International Conference on Information and Communication Systems, ICICS, IEEE, 2020, pp. 243–248.
- [64] M. Siavvas, E. Gelenbe, D. Kehagias, D. Tzovaras, Static analysis-based approaches for secure software development, in: Security in Computer and Information Sciences: First International ISCIS Security Workshop 2018, EuroCYBERSEC 2018, London, UK, February 26–27, 2018, Revised Selected Papers 1, Springer International Publishing, 2018, pp. 142–157.
- [65] S. Moshtari, A. Sami, M. Azimi, Using complexity metrics to improve software security, Comput. Fraud Secur. 2013 (5) (2013) 8–17.
- [66] S. Moshtari, A. Sami, Evaluating and comparing complexity, coupling and a new proposed set of coupling metrics in cross-project vulnerability prediction, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 1415–1421.
- [67] J. Walden, J. Stuckman, R. Scandariato, Predicting vulnerable components: Software metrics vs text mining, in: 2014 IEEE 25th International Symposium on Software Reliability Engineering, IEEE, 2014, pp. 23–33.
- [68] I. Kaloutsoglu, M. Siavvas, D. Tsoukalas, D. Kehagias, Cross-project vulnerability prediction based on software metrics and deep learning, in: Computational Science and Its Applications-ICCSA 2020: 20th International Conference, Cagliari, Italy, July 1–4, 2020, Proceedings, Part IV 20, Springer, 2020, pp. 877–893.
- [69] Z. Yu, N.A. Kraft, T. Menzies, Finding better active learners for faster literature reviews, Empir. Softw. Eng. 23 (2018) 3161–3186.
- [70] Z. Yu, C. Theisen, L. Williams, T. Menzies, Improving vulnerability inspection efficiency using active learning, IEEE Trans. Softw. Eng. 47 (11) (2019) 2401–2420.
- [71] Q. Hu, Y. Guo, X. Xie, M. Cordy, L. Ma, M. Papadakis, Y. Le Traon, Active code learning: Benchmarking sample-efficient training of code models, IEEE Trans. Softw. Eng. (2024).
- [72] H.J. Kang, D. Lo, Active learning of discriminative subgraph patterns for api misuse detection, IEEE Trans. Softw. Eng. 48 (8) (2021) 2761–2783.
- [73] Y. Zhou, H. Jin, X. Yang, T. Chen, K. Narasimhan, H.C. Gall, BRAID: an API recommender supporting implicit user feedback, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021, pp. 1510–1514.
- [74] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning, ACM Trans. Intell. Syst. Technol. (2019) 1–19.
- [75] Y. Yang, X. Hu, Z. Gao, J. Chen, C. Ni, X. Xia, D. Lo, Federated learning for software engineering: A case study of code clone detection and defect prediction, IEEE Trans. Softw. Eng. 50 (2) (2024) 296–321.

Zhidan Yuan received the M.D. degree in computer technology from Nantong University, Nantong, in 2020. He is currently pursuing the Ph.D degree at Shanghai Maritime University, Shanghai. His research interest is software engineering, in particular Vulnerability Detection, Fault Localization, and Program Repair.

Xiang Chen received the B.Sc. degree in information management and systems from Xi'an Jiaotong University, China in 2002. Then he received his M.Sc., and Ph.D. degrees in computer software and theory from Nanjing University, China in 2008 and 2011 respectively. He is currently an Associate Professor at the School of Artificial Intelligence and Computer Science, Nantong University. He has authored or co-authored more than 160 papers in refereed journals or conferences, such as IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, Empirical Software Engineering, Information and Software Technology, Journal of Systems and Software, Software Testing, Verification and Reliability, Journal of Software: Evolution and Process, International Conference on Software Engineering (ICSE), International Conference on the Foundations of Software Engineering (FSE), International Symposium on Software Testing and Analysis (ISSTA), International Conference Automated Software Engineering (ASE), International Conference on Software Maintenance and Evolution (ICSME), International Conference on Program Comprehension (ICPC), and International Conference on Software Analysis, Evolution and Reengineering (SANER). His research interests include software engineering, in particular software testing and maintenance, software repository mining, and empirical software engineering. He received two ACM SIGSOFT distinguished paper awards in ICSE 2021 and ICPC 2023. He is the editorial board member of Information and Software Technology. More information can be found at:

<https://xchencs.github.io/index.html>.

Juan Zhang received the Ph.D. degree in forestry equipment engineering from Beijing Forestry University, Beijing, China, in 2011. She is currently a full professor and the dean of the College of Information Engineering at Jiangsu Maritime Institute. Her research interests include artificial intelligence and image processing and analysis.

Weiming Zeng received the Ph.D. degree in biomedical engineering from Southeast University, Nanjing, China, in 2005. He is currently a Full Professor with the College of Information Engineering, Shanghai Maritime University, Shanghai, China. His research interests include image processing and analyzing, functional connectivity, software engineering and neuroscience computation.