Expert Systems With Applications 291 (2025) 128504

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

On the value of imbalance loss functions in enhancing deep learning-based vulnerability detection

Xiaoxue Ma [a], Yanzhong He [b,c], Jacky Keung [d], Cheng Tan [c], Chuanxiang Ma [e,f], Wenhua Hu [c], Fuyang Li [c,*]

[a] School of Science and Technology, Hong Kong Metropolitan University, Hong Kong, China
[b] School of Computer Science, Wuhan University, Wuhan, Hubei, China
[c] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, Hubei, China
[d] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[e] School of Computer Science, Hubei University, Wuhan, Hubei, China
[f] Hubei Key Laboratory of Big Data Intelligent Analysis and Application (Hubei University), Wuhan, Hubei, China

ARTICLE INFO

Keywords:
Vulnerability detection
Deep learning
Imbalance loss functions
Empirical study

ABSTRACT

Software vulnerability detection is crucial in software engineering and information security, and deep learning has been demonstrated to be effective in this domain. However, the class imbalance issue, where non-vulnerable code snippets vastly outnumber vulnerable ones, hinders the performance of deep learning-based vulnerability detection (DLVD) models. Although some recent research has explored the use of imbalance loss functions to address this issue and enhance model efficacy, they have primarily focused on a limited selection of imbalance loss functions, leaving many others unexplored. Therefore, their conclusions about the most effective imbalance loss function may be biased and inconclusive. To fill this gap, we first conduct a comprehensive literature review of 119 DLVD studies, focusing on the loss functions used by these models. We then assess the effectiveness of nine imbalance loss functions alongside cross entropy (CE) loss (the standard balanced loss function) on two DLVD models across four public vulnerability datasets. Our evaluation incorporates six performance metrics, with results analyzed using the Scott-Knott effect size difference (ESD) test. Furthermore, we employ interpretable analysis to elucidate the impact of loss functions on model performance. Our findings provide key insights for DLVD, which mainly include the following: the LineVul model consistently outperforms the ReVeal model; label distribution aware margin (LDAM) loss achieves the highest Precision, while logit adjustment (LA) loss yields the best Recall; Class balanced focal (CB-Focal) loss excels in comprehensive performance on extremely imbalanced datasets; and LA loss is optimal for nearly balanced datasets. We recommend using LineVul with either CB-Focal loss or LA loss to enhance DLVD outcomes. Our source code and datasets are available at https://github.com/YanzhongHe/DLVD-ImbalanceLossEmpirical.

1. Introduction

Software vulnerabilities are structural or design flaws within software applications that could be exploited by attackers to compromise the security, functionality, and integrity of the system, network, or data with which the software interacts (Fu & Tantithamthavorn, 2022; Yu, Lin, Hu, Keung, & Xia, 2025). When software vulnerabilities go undetected, the potential consequences can be catastrophic (Steenhoek, Gao, & Le, 2024). A recent example is the Change Healthcare attack[1] in February 2024, where the ransomware group Blackcat exploited vulnerabilities in the company's IT systems, causing widespread disruptions to healthcare services in the United States. The attack forced the company to shut down critical systems, affecting hospitals, pharmacies, and medical offices, and resulted in the theft of sensitive patient data. This incident highlights the critical importance of vulnerability detection, a task that has been extensively studied in the fields of software engineering and information security (Aftabi, Moradi, Mahroo, & Kianfar, 2025; Alabdulwahab, Cheong, Seo, Kim, & Son, 2024; Liu et al., 2024a; Tang, Tang, Gui, Hu, & Zhao, 2024; Tian, Tian, Lv, Chen, & Chen, 2024).

In recent years, the widespread application of deep learning technology has led researchers to propose numerous deep learning-based vulnerability detection (DLVD) approaches for automatically extracting

* Corresponding author.
E-mail addresses: kxma@hkmu.edu.hk (X. Ma), heyanzhong@whut.edu.cn (Y. He), Jacky.Keung@cityu.edu.hk (J. Keung), cheng_tan@whut.edu.cn (C. Tan), mcx838@hubu.edu.cn (C. Ma), whu10@whut.edu.cn (W. Hu), fyli@whut.edu.cn (F. Li).

[1] https://cn-sec.com/archives/3589605.html

https://doi.org/10.1016/j.eswa.2025.128504
Received 16 January 2025; Received in revised form 20 May 2025; Accepted 4 June 2025
Available online 16 June 2025

vulnerability features from code snippets, achieving remarkable success in vulnerability detection (Croft, Babar, & Kholoosi, 2023; Liu et al., 2024b; Qiu et al., 2024; Steenhoek et al., 2024; Zhang, Liu, Hu, Xia, & Li, 2023b). However, a significant practical challenge in DLVD that cannot be ignored is the class imbalance, where the number of non-vulnerable code snippets in the training set significantly exceeds that of vulnerable ones. Specifically, the commonly used vulnerability datasets such as Reveal (Chakraborty, Krishna, Ding, & Ray, 2022) and Bigvul (Fan, Li, Wang, & Nguyen, 2020) are highly imbalanced, with the vulnerability percentages of 9.85 % and 6.17 %, respectively. Chakraborty et al. discovered that DLVD models of ten exhibit a bias toward the majority class (i.e., non-vulnerable code snippets) when trained on imbalanced datasets, leading to poor recall (Chakraborty et al., 2022). The low recall indicates that DLVD models may overlook or fail to effectively identify a significant portion of vulnerable code snippets, thereby limiting their detection efficacy.

1.1. Motivations

In order to alleviate the problem of DLVD model performance degradation caused by class imbalance, some studies (Hanif & Maffeis, 2022; Islam, Torre, Manuel, Bou-Harb, & Najafirad, 2023; Lin, Jia, & Wu, 2022; Lin et al., 2021, 2019; Liu et al., 2024a; Nguyen et al., 2020; Song, Wang, Yang, & Wang, 2023; Steenhoek et al., 2024; Tang et al., 2024; Tang, Tang, Ban, Zhao, & Feng, 2023; Wei, Lin, Li, & Jia, 2021; Wen et al., 2023) have proposed using imbalance loss functions to replace the most common loss function cross entropy (CE) loss. By adjusting probability or assigning higher loss weights to vulnerable code snippets, these imbalance loss functions make the DLVD models pay more attention to vulnerable code snippets, thereby improving the detection performance of these code snippets. For instance, to alleviate class imbalance, ten studies (Hanif & Maffeis, 2022; Lin et al., 2022, 2021, 2019; Nguyen et al., 2020; Steenhoek et al., 2024; Tang et al., 2024, 2023; Wei et al., 2021; Wen et al., 2023) all applied weighted cross entropy (WCE) loss to train the DLVD model. Moreover, Islam et al. (2023) and Song et al. (2023) trained DLVD models by incorporating Focal loss and class balanced focal (CB-Focal) loss, respectively. These studies demonstrate the performance improvements of DLVD models when using imbalance loss function compared to CE loss. However, each study focuses on one single imbalance loss function, and many other imbalance loss functions have not been investigated. Therefore, it remains unclear which imbalance loss function is more effective in mitigating the class imbalance problem. As far as we know, only Liu et al. (2024a) developed a gated graph neural network (GGNN) vulnerability detection model trained by Focal loss and compared the effects of various loss functions (i.e., CE loss, label distribution aware margin (LDAM) loss, gradient harmonized margin (GHM) loss, Focal loss, Dice loss, and dice coefficient (DSC) loss) on model performance across the Devign (Zhou, Liu, Siow, Du, & Liu, 2019) dataset.[2] However, it is important to note that the vulnerability percentage of the Devign dataset (i.e., 42.6 %) is not extremely imbalanced, which differs from the highly imbalanced vulnerability percentage in the real world. Additionally, this study (Liu et al., 2024a) was conducted on a single dataset and only used a specific type of DLVD model (i.e., a graph-based model). Therefore, the conclusion that Focal loss is better than the comparison loss functions is biased and still does not have enough conviction.

1.2. Our works and contributions

To address these issues, our study investigates the performance of nine imbalance loss functions compared with CE loss on two representative types of DLVD models across four widely used datasets. Specifically, to comprehensively understand the imbalance loss functions used in DLVD studies, we first conduct a literature review of 119 papers between 2017 and December 2024, focusing on the loss functions employed in these models. From this literature review, we identify only thirteen of these studies employed imbalance loss functions to address the class imbalance issue. We collect seven imbalance loss functions (i.e., LDAM loss, WCE loss, GHM loss, Focal loss, CB-Focal loss, Dice loss, and DSC loss) from the ones employed in these thirteen papers. Additionally, we incorporate two other imbalance loss functions (i.e., logit adjustment (LA) loss Menon et al., 2021 and class balanced cross entropy (CB-CE) loss Cui, Jia, Lin, Song, & Belongie, 2019) that have been proven effective in addressing the class imbalance problem in the field of artificial intelligence (Cui et al., 2019; Menon et al., 2021), resulting in a total of nine imbalance loss functions. In addition, unlike Liu et al. (2024a) who only used one graph-based DLVD model on a single dataset, we use two representative DLVD models (i.e., token-based and graph-based models) and conduct experiments on four common vulnerability datasets. Through this design, we aim to improve the generalizability of our experimental results. Subsequently, we apply six metrics and the Scott-Knott effect size difference (ESD) test (Tantithamthavorn, Hassan, & Matsumoto, 2018) to evaluate the effectiveness of imbalance loss functions. Based on the experimental results, we provide suggestions for practitioners and researchers to alleviate the class imbalance problem in vulnerability detection. To guide our study, we structure the following research questions (RQs):

RQ1: Do imbalance loss functions improve the performance of DLVD models? We apply the nine imbalance loss functions (LA loss, LDAM loss, GHM loss, Focal loss, CB-CE loss, CB-Focal loss, Dice loss, and DSC loss) and CE loss to train two DLVD models (LineVul Fu & Tantithamthavorn, 2022 and ReVeal Chakraborty et al., 2022) across four publicly available vulnerability datasets (i.e., Devign, Reveal, Bigvul, and Juliet Boland & Black, 2012). Then, we comprehensively evaluate the effects of these imbalance loss functions by using six metrics (i.e., Precision, Recall, F1-score, Matthews correlation coefficient (MCC), AUC, and H-measure) and apply the Scott-Knott ESD test to determine whether these loss functions cause statistically significant differences in model performance. **Findings:** LineVul demonstrates superior vulnerability detection performance compared to ReVeal across all four datasets. Considering all six metrics, CE loss is rarely the optimal choice for vulnerability datasets with class imbalance, as several imbalance loss functions consistently outperform CE loss in most cases. Specifically, employing the LineVul model for vulnerability detection with LDAM loss enhances Precision, while LA loss optimizes high Recall. For high overall performance (high F1-score, MCC, AUC, and H-measure), CB-Focal loss and LA loss are the preferred choices.

RQ2: What degree of orthogonality is observed among the studied loss functions in detecting vulnerable code snippets? To further explore the performance differences of loss functions, we apply orthogonality analysis to evaluate how effectively each loss function uniquely identifies vulnerable code snippets. This analysis measures the number of vulnerable code snippets correctly classified by one loss function but misclassified by all others. Venn diagrams are used to visualize the performance differences among the top seven loss functions across the testing datasets. **Findings:** The orthogonality analysis further proves that imbalance loss functions (e.g., LA loss, WCE loss, GHM loss, and DSC loss) with high Recall also demonstrate strong performance in orthogonality. For example, LA loss shows the highest Recall performance on the LineVul model across all four datasets. Through orthogonality analysis, it effectively enhances the LineVul model to identify more unique vulnerable code snippets than other loss functions.

RQ3: Why do imbalance loss functions alleviate the class imbalance problem? We use the language interpretability tool (LIT) (Tenney et al., 2020) to perform interpretability analysis on DLVD models trained with the best imbalance loss function across two extremely imbalanced datasets, identifying the important parts of vulnerable code snippets that

[2] This study used the Devign dataset, along with its subsets FFmpeg and QEMU, as the research datasets. In the field of vulnerability detection, FFmpeg and QEMU are typically combined to create the complete Devign dataset for research. Therefore, we consider this study to be a performance comparison of different loss functions conducted on one single dataset.

the models focus on. **Findings:** The interpretability analysis results show that these imbalance loss functions make the DLVD model pay more attention to the code lines that lead to vulnerabilities by adjusting the loss weight or probability, thus enhancing model performance.

The main contributions of our study are outlined as follows:

(1) We perform a comprehensive literature review of 119 DLVD studies to explore the role of loss functions in improving DLVD model performance. Our analysis identifies a significant gap: the majority of studies (approximately 89.1 %) have not adequately addressed the issue of class imbalance in DLVD, nor have they employed imbalance loss functions to mitigate its impact.

(2) To the best of our knowledge, we conduct the first extensive study systematically assessing the impact of imbalance loss functions on model performance in DLVD. We evaluate the effects of nine imbalance loss functions, along with CE loss, on two representative DLVD models, providing a comprehensive analysis across four publicly available benchmark vulnerability datasets.

(3) Based on our empirical results, we offer several recommendations for researchers and software developers in DLVD. Firstly, we advise using the LineVul model for DLVD tasks. Secondly, practitioners and researchers should consider incorporating imbalance loss functions to alleviate class imbalance issues. Finally, they should select the most suitable imbalance loss function based on task objectives: LDAM loss for high Precision, LA loss for optimizing Recall, and CB-Focal loss and LA loss for overall performance.

The work is an expanded version of our prior study published at the proceedings of the 15th Asia-Pacific Symposium on Internetware (Internetware 2024) (He et al., 2024), incorporating the following updates: We first conduct a comprehensive literature review of 119 DLVD studies and identify the loss functions used for model training. Building upon the evaluation of Devign, Reveal, and Juliet datasets in He et al. (2024), we further investigate the effectiveness of imbalance loss functions on the Bigvul dataset, incorporating additional imbalance loss functions such as WCE loss, Dice loss, and DSC loss. Moreover, we introduce one comprehensive metric (MCC) and two additional threshold-independent evaluation metrics (i.e., AUC and H-measure) to provide a more comprehensive performance assessment. Based on comprehensive experimental results, we further evaluate the performance of loss functions in DLVD through orthogonality analysis and conduct interpretability analysis to explore the internal mechanisms of DLVD models trained with imbalance loss functions. Finally, we introduce additional discussions on the key hyperparameter search for imbalance loss functions and on deeper data feature space analysis via t-SNE visualization, providing further insights into their impact on model performance and feature representations.

1.3. Organization

The remaining sections of the paper are organized as follows: Section 2 introduces the use of DLVD models and imbalance loss functions in our study. Section 3 presents the literature review. Section 4 outlines the experimental setup. Section 5 provides the experimental results. Section 6 offers the discussion of our study. Finally, Section 9 summarizes our study and discusses future work.

2. Preliminaries

DLVD automatically detects and identifies vulnerable code snippets using deep learning techniques. A code snippet can be represented as $C_i = (\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i$ is the embedding vectors of the i-th code snippet in the feature space, and y_i represents the class label (0 for non-vulnerable and 1 for vulnerable) of the i-th code snippet. The vulnerability dataset can be expressed as $D = \{C_i = (\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, where N is the total number of code snippets in D. Vulnerability detection aims to train a binary classification model from D to predict whether new code snippets contain vulnerabilities. Given an input code snippet $\boldsymbol{x}$, the DLVD model processes it to produce a predicted output vector $z = [z_0, z_1]^\top$, where z_0 and z_1 represent the DLVD model's scores for the two different classes. The loss function treats each class as mutually exclusive, calculating the probability distribution for vulnerabilities as $p = \frac{e^{z_1}}{e^{z_0}+e^{z_1}}$.

The section provides a brief overview of the two DLVD models and nine imbalance loss functions investigated in our study.

2.1. DLVD Models

Recent deep learning approaches for vulnerability detection can be divided into two main types: token-based DLVD models and graph-based DLVD models. To select the most suitable DLVD models, we take three factors into consideration. Firstly, we look for models that represent two different types of DLVD models described in Section 3. Secondly, we select models commonly used as baselines for assessing generalizability in the vulnerability detection community. Therefore, we choose the following models:

(1) **LineVul** (Fu & Tantithamthavorn, 2022) is a Transformer-based model for vulnerability detection. This model treats code snippets as sequences of tokens, leveraging Transformers to capture long-range dependencies within these sequences. By doing so, LineVul achieves a contextually rich vector representation, enhancing its capability to predict vulnerabilities accurately. Thus, we select LineVul as the representative token-based model.

(2) **ReVeal** (Chakraborty et al., 2022) is a GGNN vulnerability detection for vulnerability detection. It converts code snippets into code property graphs to capture syntactic and semantic features. It also uses a trained multi-layer perceptron-based representation learner to learn feature vector representations that maximally distinguish between positive and negative classes. Thus, we select ReVeal as the representative graph-based model.

For the training phase, LineVul and ReVeal models adopt the **CE loss** (Rubinstein, 1999), which is widely utilized in various DLVD models, including DeepVD (Wang et al., 2023b) and MVD+ (Cao et al., 2023), as well as other DLVD models in the field (Zhang, Liu, Xin, & Yao, 2023a; Zhang et al., 2023b; Zhou et al., 2019). The CE loss is formulated as follows:

$$\mathcal{L}_{\mathrm{CE}} = \frac{1}{N}\sum_{i=1}^{N} -y_i \log(p_i) - (1-y_i)\log(1-p_i) \tag{1}$$

where p_i reflects the predicted probability that the i-th code snippet is vulnerable, and $y_i \in \{0, 1\}$ represents the ground-truth label, with $y_i = 1$ denoting a vulnerable code snippet and $y_i = 0$ denoting a non-vulnerable code snippet. During the optimization process, CE loss treats each training code snippet equally. That is, it assigns the same weight to vulnerable and non-vulnerable code snippets. However, most of the datasets in the field of vulnerability detection are imbalanced (Chakraborty et al., 2022). Thus, CE loss tends to prioritize a higher proportion of non-vulnerable code snippets (i.e., the majority class), making it challenging for models to learn the characteristics of vulnerable code snippets (i.e., the minority class), which leads to high false negatives. Specifically, a high false negative indicates that DLVD models may misclassify vulnerable code snippets as non-vulnerable code snippets, resulting in poor Recall when faced with high class imbalance (Chakraborty et al., 2022).

2.2. Imbalance loss functions

In our study, we explore the performance of nine imbalance loss functions and CE loss on two DLVD models. Among the nine imbalance loss functions, most are optimized based on CE loss, and we categorize them into two groups: (1) adjusting probability (i.e., LA loss and LDAM loss) and (2) adjusting weight (i.e., WCE loss, GHM loss, Focal loss, CB-CE loss, and CB-Focal loss). For the other two imbalance loss functions, Dice loss is proposed to deal with the problem of class imbalance based on the Sørensen-Dice coefficient (Milletari, Navab, & Ahmadi, 2016).

Table 1
Summary of imbalance loss functions.

Full Name	Abbreviation	Key Characteristics
Logit adjustment loss	LA loss	Probability strategy: Adjust logits using class priors to focus on the minority class (i.e., vulnerable code snippets).
Label distribution aware margin loss	LDAM loss	Probability strategy: Adjust logits using class-dependent margin factor to focus on the minority class.
Weighted cross entropy loss	WCE loss	Weighting strategy: Introduce a weighting factor assigned to each class based on its proportion in order to give more emphasis to the minority class.
Gradient harmonized margin loss	GHM loss	Weighting strategy: Introduce the density of gradient magnitudes in order to emphasize rare and hard examples while down-weighting abundant easy ones.
Focal loss	Focal loss	Weighting strategy: Introduce the class-specific weighting factor and focusing factor to encourage the model to learn more from hard examples.
Class balanced cross entropy loss	CB-CE loss	Weighting strategy: Introduce the class-specific weighting factor based on the concept of the "effective number" of examples to focus on the minority class.
Class balanced focal loss	CB-Focal loss	Weighting strategy: Combine the ideas of Focal loss and CB-CE loss to encourage the model to focus on hard-to-classify and minority class instances.
Dice loss	Dice loss	Similarity strategy: Measure set similarity and give balanced attention to false positives and false negatives.
Dice coefficient loss	DSC loss	Similarity strategy: Integrate the focal factor from Focal loss into Dice loss, giving greater emphasis on hard examples.

DSC loss introduces the weight factor of Focal loss on the basis of Dice loss, so that the model can pay more attention to minority class samples during training, thereby effectively alleviating the impact of class imbalance (Li et al., 2020). Table 1 presents detailed descriptions of these imbalance loss functions. We introduce these loss functions in the above order as follows:

(1) **LA loss** (Menon et al., 2021) introduces label-dependent offsets by integrating class priors, allowing for the adjustment of each logit to better align with the data samples. Specifically, the class prior refers to the original probability distribution of each class within the entire dataset, reflecting the inherent imbalance in class representation. And the logit refers to the raw, unnormalized output of the neural network before applying a probability function, reflecting the model's confidence score in each class. Therefore, it ensures the model remains sensitive to the minority class (i.e., vulnerable code snippets). Consequently, the vulnerability probability $p = \frac{e^{z_1}}{e^{z_0}+e^{z_1}}$ is refined to $p_{\mathrm{LA}} = \frac{e^{z_1+\tau\cdot\log\pi_1}}{e^{z_0+\tau\cdot\log\pi_0}+e^{z_1+\tau\cdot\log\pi_1}}$. The LA loss is formulated as follows:

$$\mathcal{L}_{\mathrm{LA}} = \frac{1}{N}\sum_{i=1}^{N} -y_i \log(p_{\mathrm{LA}_i}) - (1-y_i)\log(1-p_{\mathrm{LA}_i}), \tag{2}$$

where p_{LA_i} represents the predicted probability that the i-th code snippet is vulnerable, π signifies estimates of class priors, and τ is a temperature scaling parameter that modulates the extent of logit adjustment, thereby controlling the influence of the class prior on the model's output and facilitating the calibration of the predicted probability distribution.

(2) **LDAM loss** (Cao, Wei, Gaidon, Aréchiga, & Ma, 2019) employs the class-dependent margin factor Δ, to tackle the issue of class imbalance. This margin factor is influenced by the frequency of training labels for each class. Specifically, the margin factor $\Delta_y = \frac{C}{n_y^{1/4}}$ is designed to adjust the decision boundary for each class (which includes both vulnerable and non-vulnerable classes), where C is a tunable hyperparameter (we use the default value $C = 0.5$ for implementation), and n_y denotes the number of samples in class y. This adjustment in the margin encourages the minority class (i.e., vulnerable code snippets) to maintain a larger distance from the decision boundary, thus improving the detection of vulnerable code snippets. Consequently, following previous research (Cao et al., 2019), the vulnerability probability $p = \frac{e^{z_1}}{e^{z_0}+e^{z_1}}$ in CE loss is refined to $p_{\mathrm{LDAM}} = \frac{e^{z_1-\Delta_1}}{e^{z_0-\Delta_0}+e^{z_1-\Delta_1}}$. The LDAM loss is formulated as follows:

$$\mathcal{L}_{\mathrm{LDAM}} = \frac{1}{N}\sum_{i=1}^{N} -y_i \log(p_{\mathrm{LDAM}_i}) - (1-y_i)\log(1-p_{\mathrm{LDAM}_i}), \tag{3}$$

where p_{LDAM_i} denotes the predicted probability that the i-th code snippet is vulnerable, and Δ_y represents the class-dependent margin for the relevant class.

(3) **WCE loss** modifies the standard CE loss by introducing a weighting factor (ω) for each class:

$$\mathcal{L}_{\mathrm{WCE}} = \frac{1}{N}\sum_{i=1}^{N} -\omega_1 y_i \log(p_i) - \omega_0(1-y_i)\log(1-p_i), \tag{4}$$

where ω is a weight assigned to each class based on its proportion, giving more emphasis to the minority class (i.e., vulnerable code snippets). Specifically, the $y_i = 1$ denotes a vulnerable code snippet and $y_i = 0$ denotes a non-vulnerable code snippet. The weights ω_1 and ω_0 are defined as the inverse proportions of vulnerable and non-vulnerable code snippets in the dataset.

(4) **GHM loss** (Li, Liu, & Wang, 2019) is designed to balance the contributions of easy and hard samples by adjusting their weights based on the distribution of gradient magnitudes. In vulnerability detection, hard code snippets have large gradients (i.e., low confidence predictions), while easy ones have small gradients. The GHM loss assigns smaller weights to samples with high gradient density and larger weights to those with low density, thus emphasizing rare and informative instances. Let $g_i = |p_i - y_i|$ denote the gradient magnitude of the i-th sample, where $p_i \in (0,1)$ is the predicted probability and $y_i \in \{0,1\}$ is the ground truth label. The gradient density $GD(g_i)$ is calculated by counting the number of samples with gradient magnitudes within a small range ϵ around g_i, normalized by the valid length $l_\epsilon(g_i)$ of that range. Formally,

$$\delta_\epsilon(x,y) = \begin{cases} 1 & \text{if } y - \frac{\epsilon}{2} \le x < y + \frac{\epsilon}{2}, \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

$$l_\epsilon(g) = \min\left(g + \frac{\epsilon}{2}, 1\right) - \max\left(g - \frac{\epsilon}{2}, 0\right), \tag{6}$$

$$GD(g) = \frac{1}{l_\epsilon(g)}\sum_{k=1}^{N} \delta_\epsilon(g_k, g), \tag{7}$$

where g_k is the gradient magnitude of the k-th sample. The final loss is defined as:

$$\mathcal{L}_{\mathrm{GHM}} = \sum_{i=1}^{N} -\frac{1}{GD_1(g_i)} y_i \log(p_i) - \frac{1}{GD_0(g_i)}(1-y_i)\log(1-p_i), \tag{8}$$

where $\frac{1}{GD(g_i)}$ is the gradient density-based weight that dynamically scales the CE loss for each sample. Specifically, when $y_i = 1$, the gradient density-based weight for the code snippet is defined as $\frac{1}{GD_1(g_i)}$, whereas for $y_i = 0$, the gradient density-based weight is defined as $\frac{1}{GD_0(g_i)}$.

(5) **Focal loss** (Lin, Goyal, Girshick, He, & Dollár, 2017b) is designed to address the issue of class imbalance by dynamically scaling the standard CE loss. In vulnerability detection, the majority of training samples are typically non-vulnerable code snippets, which are easier to classify and dominate the loss computation, thereby hindering effective learning for the minority class (vulnerable code snippets). Focal loss mitigates this by down-weighting the loss assigned to well-classified examples and focusing more on hard, misclassified ones.

The loss function is formulated as:

$$\mathcal{L}_{\text{FL}} = \frac{1}{N}\sum_{i=1}^{N} -\alpha_1(1-p_i)^{\gamma} y_i \log(p_i) - \alpha_0 p_i^{\gamma}(1-y_i)\log(1-p_i), \tag{9}$$

where α_1 and α_0 are the class-specific weighting factors corresponding to the proportions of non-vulnerable and vulnerable code snippets in the dataset, respectively. This ensures that the minority class (vulnerable code snippets) receives a proportionally higher weight. The focusing parameter γ adjusts the rate at which easy examples are down-weighted: a larger γ places more emphasis on hard, misclassified samples by reducing the loss contribution from correctly classified ones. Specifically, $(1-p_i)^{\gamma}$ and p_i^{γ} serve to reduce the impact of easy examples from each class, enhancing the model's ability to learn from difficult samples and improving detection performance under severe class imbalance. By adjusting α and γ, Focal Loss can effectively improve model performance under severe class imbalance by emphasizing learning on underrepresented samples.

(6) **CB-CE loss** (Cui et al., 2019) addresses the problem of class imbalance by introducing a weighting strategy based on the concept of the "effective number" of samples. In vulnerability detection tasks, where vulnerable code snippets are significantly underrepresented, conventional CE loss tends to be biased toward the majority class. CB-CE loss reduces this bias by assigning lower weights to overrepresented classes and higher weights to underrepresented ones.

The effective number of samples for class y is defined as:

$$E_{n_y} = \sum_{j=1}^{n_y} \beta^{j-1} = \frac{1-\beta^{n_y}}{1-\beta}, \tag{10}$$

where n_y denotes the number of code snippets belonging to class y, and $\beta \in [0,1)$ controls the rate at which additional samples contribute less to the effective number. Specifically, the j^{th} sample contributes β^{j-1}, implying that as j increases, the marginal information gain from new samples diminishes due to redundancy.

By applying a class-specific weighting factor $\frac{1}{E_{n_y}}$ to the standard CE loss, the final CB-CE loss is formulated as:

$$\mathcal{L}_{\text{CBCE}} = \frac{1}{N}\sum_{i=1}^{N} -\frac{1}{E_{n_1}} y_i \log(p_i) - \frac{1}{E_{n_0}}(1-y_i)\log(1-p_i). \tag{11}$$

This reweighting mechanism ensures that each class contributes equally to the loss, thereby mitigating the adverse effects of class imbalance and enhancing model performance, especially on the minority class.

(7) **CB-Focal loss** (Cui et al., 2019) combines the ideas of class-balanced weighting and Focal loss to simultaneously address the challenges of class imbalance and the dominance of easy samples in model training. In vulnerability detection, where non-vulnerable code snippets typically outnumber vulnerable ones, CB-Focal loss encourages the model to focus on hard-to-classify and minority-class instances (vulnerable code snippets).

The loss introduces the effective number of samples E_{n_y} as a replacement for the class weight factor α in Focal Loss. Specifically, the contribution of each sample is modulated both by its difficulty (using the focusing term $(1-p_i)^{\gamma}$ or p_i^{γ}) and the inverse of the effective number of samples, yielding the following loss function:

$$\mathcal{L}_{\text{CBFL}} = \frac{1}{N}\sum_{i=1}^{N} -\frac{1}{E_{n_1}}(1-p_i)^{\gamma} y_i \log(p_i) - \frac{1}{E_{n_0}} p_i^{\gamma}(1-y_i)\log(1-p_i). \tag{12}$$

(8) **Dice loss** (Li et al., 2020). Based on the DSC, this loss function measures similarity between two sets and effectively handles class imbalances in machine learning tasks. It is particularly useful when positive examples are rare, ensuring equal consideration of false positives and false negatives. For an individual sample x_i, the corresponding DSC is defined as:

$$\text{DSC}(x_i) = \frac{2p_i y_i + \gamma}{p_i + y_i + \gamma}, \tag{13}$$

where p_i is the predicted probability that the i-th sample is vulnerable, and y_i is the ground truth label. The DSC quantifies the overlap between the predicted and true labels for each sample, emphasizing the importance of correct positive predictions.

The Dice loss is then defined as follows:

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2\sum_{i=1}^{N} p_i y_i + \gamma}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} y_i^2 + \gamma}, \tag{14}$$

where p_i reflects the predicted probability that the i-th code snippet is vulnerable, y_i is the ground truth label, and γ is a smoothing factor added to avoid division by zero, set as 1 following prior research (Li et al., 2020). This formulation uses squared sums in the denominator following (Li et al., 2020), which provides a smoothed variant of Dice loss that improves gradient stability during training.

(9) **DSC loss** (Li et al., 2020) extends the traditional Dice loss by incorporating the focal factor from Focal loss, emphasizing harder-to-classify examples and paying more attention to the difficulty of the code snippets. It is particularly effective in handling class imbalances by dynamically adjusting the weight of each example to focus more on challenging cases while reducing the impact of easy-negative examples.

For an individual sample x_i, the corresponding adaptive variant of the DSC is defined as:

$$\text{DSC}(x_i) = \frac{2(1-p_i)^{\alpha} p_i \cdot y_i + \gamma}{(1-p_i)^{\alpha} p_i + y_i + \gamma}. \tag{15}$$

Thus, the DSC loss is formulated as follows:

$$\mathcal{L}_{\text{DSC}} = 1 - \frac{2\sum_{i=1}^{N}(1-p_i)^{\alpha} p_i y_i + \gamma}{\sum_{i=1}^{N}(1-p_i)^{\alpha} p_i + \sum_{i=1}^{N} y_i + \gamma}, \tag{16}$$

where $(1-p_i)^{\alpha}$ acts as a focal factor to adjust the weight of each training example based on its difficulty, the focusing parameter α adjusts the rate at which easy examples are down-weighted. p_i reflects the predicted probability that the i-th code snippet is vulnerable, y_i is the ground truth label, and γ is a smoothing factor to avoid division by zero. Following previous study (Li et al., 2020), we set $\alpha = 1$ and $\gamma = 1$.

3. Literature review

To explore the advancements in applying imbalance loss functions in DLVD, we conduct a literature search following the methodology outlined by Zhou et al. (2018). We focus on publications from 2017 to 2024,[3] starting in 2017, because this marks the publication of the first DLVD paper (Lin, Zhang, Luo, Pan, & Xiang, 2017a) (abbreviated as "POSTER 2017 CCS paper" in the following). Following the practice of the previous studies (Yu et al., 2022; Zhang et al., 2024; Zhou et al., 2018), our search strategy comprises two phases: the search and the screening, as illustrated in Fig. 1. In the *search* phase, we establish the following five-point inclusion criteria: (1) the study proposes a novel DLVD model; (2) the study is written in English; (3) the full text is available; (4) if the study exists in both conference and journal versions, only the journal version is included; and (5) the prediction scenario is to classify code snippets into vulnerable code snippets and non-vulnerable code snippets.

Search Phase: We conduct a literature search using sources, including Google Scholar, IEEE Xplore Digital Library, ScienceDirect, and ACM

[3] The literature search was conducted on December 2024.

Table 2
Systematic literature reviews on imbalance loss functions for DLVD Models.

Study	Year	Venue	Imbalance Loss Function
Lin et al. (2019)	2019	IEEE Transactions on Dependable and Secure Computing	WCE loss
Nguyen et al. (2020)	2020	Pacific-Asia Conference on Knowledge Discovery and Data Mining	WCE loss
Lin et al. (2021)	2021	Neural Computing and Applications	WCE loss
Wei et al. (2021)	2021	Algorithms	WCE loss
Hanif and Maffeis (2022)	2022	International Joint Conference on Neural Networks	WCE loss
Lin et al. (2022)	2022	Mathematics	WCE loss
Tang et al. (2023)	2023	Journal of Systems and Software	WCE loss
Wen et al. (2023)	2023	International Conference on Software Engineering	WCE loss
Islam et al. (2023)	2023	European Symposium on Security and Privacy	Focal loss
Song et al. (2023)	2023	Information and Software Technology	CB-Focal loss
Liu et al. (2024a)	2024	Expert Systems with Applications	Focal loss Dice loss DSC loss GHM loss LDAM loss
Tang et al. (2024)	2024	Expert Systems with Applications	WCE loss
Steenhoek et al. (2024)	2024	International Conference on Software Engineering	WCE loss

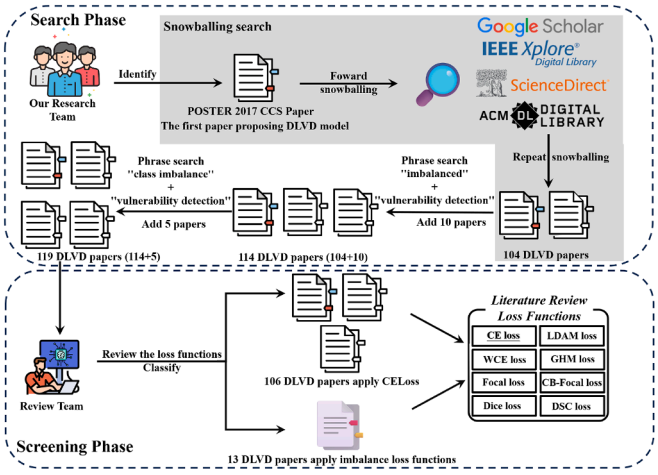


Fig. 1. The literature review process for our study.

Digital Library. We select Google Scholar as the primary search tool because it provides a simple way to broadly search for scholarly literature.[4] First, we perform a forward snowballing search (Wohlin, 2014) by recursively checking the citations of the "POSTER 2017 CCS paper". This process begins by identifying all literature that cites the POSTER 2017 CCS paper (for example, P_1, P_2, …, P_{n-1}, P_n). Next, we apply the previously defined inclusion criteria to these n papers to select those proposing DLVD models. This process is then repeated for the selected papers until no further studies meeting the criteria are found. Ultimately, this forward-snowballing search yields 104 relevant articles. Next, we conduct searches using specific phrases to identify other undiscovered DLVD papers. Similarly, we apply the same five-point inclusion criteria to the papers found in this process. Using the terms "imbalanced" + "vulnerability detection", we discover an additional 10 relevant articles (Hanif & Maffeis, 2022; Lin et al., 2022, 2021, 2019; Nguyen et al., 2020; Steenhoek et al., 2024; Tang et al., 2024, 2023; Wei et al., 2021; Wen et al., 2023). We then use "class imbalance" + "vulnerability detection" as our search term, leading to the identification of 5 more relevant articles (Cao et al., 2023; Dong, Tang, Cheng, Yang, & Wang, 2023; Wang et al., 2024b; Wen, Gao, Gao, Xiao, & Lyu, 2024; Xu et al., 2024). Finally, we explore the phrases "imbalanced" + "vulnerability prediction" and "class imbalance" + "vulnerability prediction", but do not find any additional relevant studies. Through these steps, we identify a total of 119 articles (104 from forward snowballing + 15 from phrase searches) that propose new DLVD models for vulnerability detection. Thus, our comprehensive search results in 119 DLVD articles in the search phase.

Screening Phase: We thoroughly review the loss functions used for training DLVD models in the 119 selected papers. We observe that only 13 of these studies employed imbalance loss functions to mitigate the impact of class imbalance on model performance. Table 2 summarizes the imbalance loss functions used in these 13 papers, specifically detailing seven loss functions: LDAM loss, WCE loss, GHM loss, Focal loss, CB-Focal loss, Dice loss, and DSC loss.

3.1. Deep learning-based vulnerability detection with balance loss function

As deep learning continues to be widely used in the field of software engineering, many studies have employed deep learning techniques for software vulnerability detection. Current DLVD models can be classified into two types: token-based DLVD models and graph-based DLVD models. Token-based models treat code snippets as sequences of tokens, which are represented as vectors using text embedding techniques. These models then employ various neural networks to automatically capture vulnerability features from these vectorized representations (Fu & Tantithamthavorn, 2022; Li et al., 2018; Russell et al., 2018). For instance, Russell et al. (2018) effectively extracted vulnerability features from code token sequences using a convolutional neural network to identify hierarchical features and a recurrent neural network to capture sequential dependencies. VulDeePecker (Li et al., 2018) and SySeVR (Li et al., 2021b) utilized long short-term memory and gated recurrent unit models while disregarding less critical code. They enhanced vulnerability detection performance by focusing on code slices extracted from specific "interesting points", such as array indexing, pointer usage, and API calls. Moreover, Fu and Tantithamthavorn (2022) proposed LineVul, a BERT architecture with self-attention layers, which can extract the long-term dependency relationship of code token sequences. Qiu et al. (2024) proposed MGVD, a novel DLVD model for detecting function-level vulnerabilities. It uses the Byte Pair Encoding tokenizer to process statement text and the code text of each graph into tokens, which are then converted into feature vectors with Sent2Vec, enhancing detection vulnerability performance. Jiang, Zhang, Su, Treude, and Wang (2024) introduced StagedVulBERT, a novel vulnerability detection framework that combines a pre-trained code language model with a coarse-to-fine strategy. They proposed CodeBERT-HLS, a component that captures semantics at both the token and statement levels, enhancing multi-granular vulnerability detection and efficiently handling longer code sequences.

In contrast, graph-based DLVD models (Chakraborty et al., 2022; Li, Wang, & Nguyen, 2021a; Zhou et al., 2019) convert code snippets into various types of graphs to analyze their structural characteristics.

[4] https://scholar.google.com/intl/en/scholar/about.html

These models then utilize different graph neural networks to capture and learn these structural features, enhancing the effectiveness of vulnerability detection. For example, Zhou et al. (2019) proposed the Devign model, which automates the process of learning vulnerability patterns from code property graphs using graph neural networks. Cheng, Wang, Hua, Xu, and Sui (2021) proposed DeepWukong, a model that leverages graph neural networks to embed code snippets into a compact low-dimensional representation. It preserves the high-level control and data flow information of code snippets without manual rule definition. Wang et al. (2023b) introduced DeepVD, a graph neural network model designed to extract class-separation information between non-vulnerable and vulnerable code snippets. Wen et al. (2023) proposed the AMPLE model that utilizes enhanced graph representation learning and graph simplification techniques to capture the global information of vulnerability code snippets effectively. Zhang et al. (2023b) employed a convolutional neural network and CodeBERT to capture path representations derived from the control flow graph. Cao et al. (2023) proposed MVD+, using a novel flow-sensitive graph neural network to capture both the semantic and syntactic features of vulnerable code snippets through abstract syntax tree and system dependence graph. Liu et al. (2024b) proposed FGVulDet, which adopts an edge-aware GGNN by combining edge type and node features to enhance the performance of vulnerability detection. Wang et al. (2024a) proposed the GCL4SVD model, which enhances vulnerability detection by converting code snippets into code graphs and adopting data denoising techniques. This method leverages the power of GGNN to achieve higher detection performance.

However, these models generally apply CE loss as the loss function, which have some weaknesses. Specifically, CE loss assigns equal importance to vulnerable and non-vulnerable code snippets during model training. Thus, when dealing with class-imbalanced data, the model tends to focus on learning from the non-vulnerable code snippets while failing to fully capture the characteristics of vulnerable code snippets. Consequently, the effectiveness of DLVD models using CE loss is compromised when applied to class-imbalanced datasets (Chakraborty et al., 2022).

3.2. Deep learning-based vulnerability detection with imbalance loss function

Table 2 provides an overview of the critical literature on training DLVD models with imbalance loss functions. The first and second columns list the research articles and their publication years, respectively, while the third and fourth columns specify the venues and the imbalance loss functions used. Ten studies (Hanif & Maffeis, 2022; Lin et al., 2022, 2021, 2019; Nguyen et al., 2020; Steenhoek et al., 2024; Tang et al., 2024, 2023; Wei et al., 2021; Wen et al., 2023) employed WCE loss to alleviate class imbalance during the training process of DLVD models. WCE loss remains widely adopted in vulnerability detection due to its conceptual simplicity, ease of implementation, and demonstrated effectiveness in handling imbalanced data distributions. Song et al. (2023) proposed HGIVul, a model for vulnerability detection in source code based on hypergraph convolution, using CB-Focal loss to alleviate the learning bias caused by the severe imbalance between vulnerable and non-vulnerable code snippets. Islam et al. (2023) introduced a method for training vulnerability detection models based on semantic graphs of source code, incorporating Focal loss to reduce the bias induced by imbalanced data. Experimental results demonstrated that this model excelled in vulnerability detection with a lower false positive rate. However, as each study focuses on one single imbalance loss function, it remains unclear which imbalance loss function is more effective in mitigating the class imbalance problem.

Based on our knowledge, only Liu et al. (2024a) developed a vulnerability detection model based on GGNN and compared the effects of various loss functions (i.e., CE loss, Focal loss, Dice loss, DSC loss, GHM loss, and LDAM loss) on model performance, finding that the model utilizing Focal loss achieved the best overall performance. However, it is important to note that Liu et al. (2024a) applied the imbalance loss functions to train one single type of DLVD model (i.e., the graph-based model) on three datasets, namely FFmpeg, QEMU, and FFmpeg + QEMU (Devign). In the field of vulnerability detection, FFmpeg and QEMU are typically combined to create the complete Devign dataset for research. Moreover, the vulnerability percentage of the three vulnerability dataset are not extremely imbalanced (i.e., 51.0 % for FFmpeg, 42.6 % for QEMU, and 45.6 % for Devign), which cannot reflect the challenges of extremely imbalanced datasets commonly faced in the actual vulnerability detection task. Therefore, these limitations raise questions about the generalizability of its performance on the other type of DLVD model (i.e., the token-based model) across other extremely imbalanced vulnerability datasets in the real world.

Given these observations, a systematic exploration of the impact of imbalance loss functions in DLVD is essential. Our research aims to systematically evaluate the impact of imbalance loss functions on the performance of DLVD models. By adopting more imbalance loss functions and applying them to train two representative types of DLVD models on highly imbalanced datasets and closely balanced datasets, we intend to provide comprehensive insights for researchers and practitioners in the field of vulnerability detection.

4. Experimental setup

In this section, we introduce the experimental procedure, the datasets studied, the evaluation metrics used, the implementation, and the statistical tests employed for result evaluation.

4.1. Experimental procedure

Fig. 2 shows the framework of our empirical study. It operates in two phases: the training phase and the testing phase. During the training phase, we input the training code snippets into DLVD models for training. Specifically, LineVul converts code snippets into sequences, which are then transformed into word and positional encoding vectors. These two vectors are combined to generate the final embedding vectors through word and position encoding. ReVeal converts code snippets into code property graphs and applies the GGNN to assign a GRU to each vertex in the graph to generate GGNN node embedding. These node embeddings are then aggregated to form the graph embedding. The generated embedding vectors are used in the subsequent model training phase. In this phase, we use nine imbalance loss functions and CE loss as the loss function to train these two DLVD models. In detail, the loss functions calculate the prediction error by measuring the discrepancy between the model's predicted labels and the ground truth labels using the validation code snippets. Based on this error, the model parameters

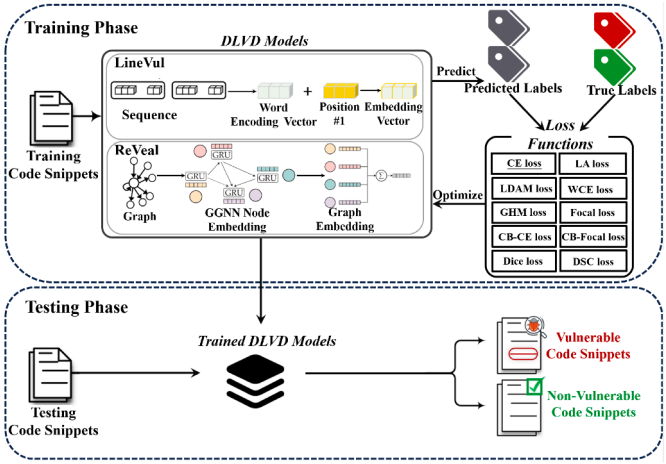


Fig. 2. The experimental framework for our study.

Table 3
The statistical information of our experimental datasets.

Dataset	# Code Snippets	Source of Annotation	Granularity	FD-LineVul	FD-ReVeal	Vulnerable Percentage
Devign	24,491	Developer-provided	Function	768	128	45.72 %
Reveal	22,734	Developer-provided	Function	768	128	9.85 %
Bigvul	15,663	Security-vendor-provided	Function	768	128	6.17 %
Juliet	38,234	Synthesized-created	Function	768	128	40.50 %

are optimized to minimize the prediction loss, thereby improving the performance of DLVD models. In the testing phase, we input the testing code snippets into the trained DLVD models for prediction, and the models output the prediction results as either vulnerable or non-vulnerable code snippets.

4.2. Datasets

In our study, we experiment on four popular datasets called Devign (Zhou et al., 2019), Reveal (Chakraborty et al., 2022),[5] Bigvul (Fan et al., 2020), and Juliet (Boland & Black, 2012). These datasets are at the function level and have been widely used in DLVD researches (Cao et al., 2023; Wen et al., 2023). However, Croft et al. (2023) identified duplicate and conflicting code snippets exist in the four vulnerability datasets. These issues could influence DLVD model training and lead to compromised benchmark effectiveness. To mitigate these problems, we apply the cleaned versions of Devign, Bigvul, and Juliet provided by Croft et al. (2023). We retain the original Reveal dataset because it lacks the necessary metadata for effective cleaning. In this study, the LineVul model utilizes the original code-label dataset, which consists of code snippets in textual format. In contrast, the ReVeal model employs a graph-based dataset derived from the textual code snippets. Specifically, following the methodology proposed by Chakraborty et al. (2022), we process the raw code snippets using the Joern tool[6] to extract their corresponding code property graphs, which are then used for training graph-based neural models. The detailed data processing procedure for the ReVeal model can be found in the work.[7] Overall, our data preprocessing steps strictly follow the original design and implementation guidelines of the LineVul and ReVeal models to ensure consistency across experiments. In our experiments, the four datasets (Devign, Reveal, Bigvul, and Juliet)[8] are used to train two DLVD models: LineVul and ReVeal. The feature dimensions extracted during the training process are 768 for LineVul and 128 for ReVeal, which are referred to as FD-LineVul and FD-ReVeal, respectively, in Table 3.

Table 3 summarizes the statistics of the datasets. The details of the four datasets are as follows:

(1) **Devign.** The Devign dataset, provided by Zhou et al. (2019), is sourced from two open-source C projects: FFmpeg and Qemu. The vulnerability labels in this dataset are provided by developers. Devign is a nearly balanced dataset. It comprises 11,197 vulnerable functions and 13,294 non-vulnerable functions, with vulnerabilities accounting for 45.72 %.

(2) **Reveal.** The Reveal dataset, provided by Chakraborty et al. (2022), originates from two open-source programs: Debian and Chromium. The vulnerability labels in this dataset are also provided by developers. Reveal is an imbalanced dataset, containing 2239 vulnerable code snippets and 20,495 non-vulnerable code snippets, resulting in a proportion of vulnerable functions of 9.85 %.

(3) **Bigvul.** The Bigvul dataset, provided by Fan et al. (2020), is sourced from over 300 open-source C/C++ projects on GitHub and encompasses 91 distinct vulnerability types listed in the common vulnerabilities and exposures datasets from 2002 to 2019. The source vulnerability labels in the Bigvul dataset are provided by security vendors. Since the Bigvul dataset is large, full training can be very time-consuming. To reduce training time, we extract 10 % of it to create the Bigvul dataset, maintaining the ratio of vulnerable to non-vulnerable code snippets for our study. The Bigvul dataset is extremely imbalanced, comprising 966 vulnerable functions and 14,697 non-vulnerable functions, with vulnerabilities accounting for 6.17 %.

(4) **Juliet.** The Juliet dataset, provided by Boland and Black (2012), is sourced from C/C++ programs with known flaws that are publicly available. The vulnerability labels in the Juliet dataset are synthesized. It is a nearly balanced dataset, containing 15,485 vulnerable code snippets and 22,749 non-vulnerable code snippets, with a vulnerable percentage of 40.50 %.

4.3. Evaluation metrics

In our study, we utilize four threshold-related evaluation metrics, namely Precision, Recall, F1-score, and MCC, along with two threshold-independent metrics, specifically the AUC and H-measure. These metrics are employed to assess the performance of the DLVD models comprehensively (Chakraborty et al., 2022; Fu & Tantithamthavorn, 2022; Nong, Ou, Pradel, Chen, & Cai, 2023; Wang, Huang, Yu, & Chen, 2023a; Wang et al., 2023b; Wen et al., 2023; Yang, Wang, Li, & Wang, 2023; Zhang et al., 2023a,b). These four evaluation metrics (i.e., Precision, Recall, F1-score, and MCC) can be calculated according to a confusion matrix, as shown in Table 4. Among them, TP represents the count of vulnerable code snippets correctly predicted as vulnerable, FN indicates the count of vulnerable ones incorrectly classified as non-vulnerable, FP represents the count of non-vulnerable code snippets incorrectly predicted as vulnerable, and TN corresponds to the count of non-vulnerable ones correctly identified as such. These metrics (i.e., Precision, Recall, F1-score, and MCC) rely directly on the values derived from this confusion matrix. For all experiments, the classification threshold is set at the default value of 0.5, with no threshold optimization performed on the validation set. Orthogonality is another metric employed to evaluate the performance of vulnerability detection models. It quantifies the number of vulnerable code snippets that are uniquely correctly classified by a specific loss function while being misclassified by all others, thereby highlighting the performance differences among models trained with different loss functions. The orthogonality metric provides a new perspective for model evaluation, revealing the performance differences between vulnerability detection models trained with different loss functions, and is an effective supplement to traditional performance indicators (such as F1-score and MCC). Therefore, we use the orthogonality metric to show

Table 4
The confusion matrix.

	Actual vulnerable	Actual non-vulnerable
Predicted vulnerable	TP	FP
Predicted non-vulnerable	FN	TN

[5] To distinguish between the vulnerability dataset called Reveal and the DLVD model also named Reveal, we refer to the dataset as Reveal and refer to the model as ReVeal in our study.

[6] https://github.com/joernio/joern

[7] https://github.com/VulDetProject/ReVeal/tree/master

[8] Our experiment datasets can be obtained through the following link: https://figshare.com/s/5483f706b04cb9a66fa3.

the performance differences between different models in vulnerability detection.

Precision: $Precision = \frac{TP}{TP+FP}$. The precision is the percentage of actual vulnerable code snippets in the dataset to all predicted vulnerable snippets. A higher Precision means DLVD models can find vulnerable code snippets more accurately.

Recall: $Recall = \frac{TP}{TP+FN}$. The recall is the percentage of the correctly predicted vulnerable code snippets to all the actual vulnerable ones in the datasets. A higher Recall means that DLVD models could capture more vulnerable code snippets.

F1-score: $F1\text{-score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$. The F1-score is viewed as the harmonic mean between Precision and Recall and is often considered the comprehensive metric (Chakraborty et al., 2022; Fu & Tantithamthavorn, 2022; Wen et al., 2023).

MCC: $MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$. The MCC provides an overview by considering all the terms of the confusion matrix, which could be used in an imbalanced environment to reduce bias (Li et al., 2023; Ma et al., 2024).

AUC refers to the area under the receiver operating characteristic (ROC) curve, which is a threshold-independent metric used to evaluate the performance of a classifier. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) across various classification thresholds. Unlike metrics (i.e., Precision, Recall, F1-score, and MCC) that depend on a specific threshold, AUC assesses the classifier's ability to distinguish between classes at all possible thresholds.

H-measure is another threshold-independent metric, particularly useful for imbalanced datasets. It takes into account the trade-off between precision and recall, as well as the costs associated with different types of errors. Unlike traditional metrics, H-measure considers the overall performance of a classifier by incorporating the decision threshold and varying loss costs for different classes, providing a more comprehensive evaluation of model performance in real-world applications.

Orthogonality, adapted from Dipongkor's study (Dipongkor & Moran, 2023), measures the independence between loss functions when applied to vulnerable code snippets. The formula for loss functions A_i, shown in Eq. 17, calculates $d(A_i)$, representing the count of correctly predicted code snippets by A_i while others misclassify them. This metric provides insights into the effectiveness of these loss functions.

$$d(A_i) = |A_{T_i} - \left(\bigcup_{j \neq i}^{n} A_{T_j} \right)|, \tag{17}$$

where T represents the set of vulnerable code snippets used to evaluate the loss functions, and A_{T_i} represents the set of vulnerable code snippets correctly classified by loss function A_i. $\bigcup_{j \neq i}^{n} A_{T_j}$ indicates the set of vulnerable code snippets correctly classified by some loss functions other than the loss function A_i. Therefore, $d(A_i)$ represents the difference between the set A_{T_i} and $\bigcup_{j \neq i}^{n} A_{T_j}$. That is, $d(A_i)$ quantifies the number of vulnerable code snippets uniquely correctly classified by loss function A_i, distinct from the classifications made by other loss functions.

4.4. Implementation

We randomly split the entire dataset into 80 % for training, 10 % for validation, and 10 % for testing, following previous works (Fu & Tantithamthavorn, 2022; Li et al., 2021a). To mitigate bias from randomness, we conduct each experiment five times. The results from these five runs are used to evaluate the performance of the studied DLVD models on each dataset with each loss function.

We use the same hyperparameter settings as in LineVul (Fu & Tantithamthavorn, 2022) and ReVeal (Chakraborty et al., 2022), including batch size, number of training epochs, and learning rate. Additionally, we select loss function parameter settings based on prior researches (Cao et al., 2019; Cui et al., 2019; Li et al., 2019; Lin et al., 2017b; Menon et al., 2021) that demonstrated optimal performance in their experiments. Specifically, we configure τ in LA loss to 1, set γ in Focal loss, and CB-Focal loss to 2. Moreover, we set π in LA loss, Δ in LDAM loss, ω in WCE loss, $GD(g)$ in GHM loss, α in Focal loss, as well as $\frac{1}{E_{n_y}}$ in CB-CE loss and CB-Focal loss based on the class imbalance proportions observed in each vulnerability dataset. During the training phase, we utilize backpropagation with the AdamW optimizer to update the DLVD models and minimize the loss function. Our experiments are performed on an Nvidia RTX 3090 24GB GPU, installed with Ubuntu 18.04, CUDA 11.3.

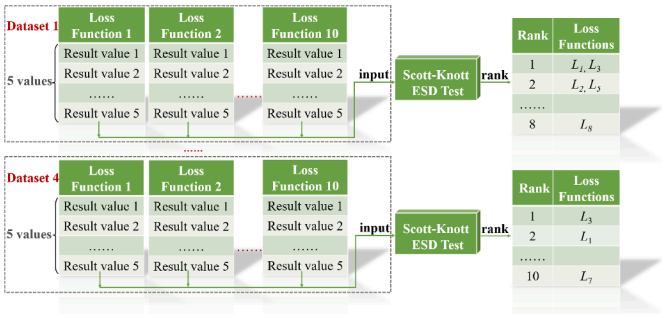


Fig. 3. The procedure of the Scott-Knott ESD test.

4.5. Statistic test

Scott-Knott ESD test (Tantithamthavorn et al., 2018) is a multiple comparison approach that uses hierarchical clustering technology to divide imbalance loss functions into different groups, with significant differences at a significance level of 0.05 ($\alpha = 0.05$). It is employed to rank imbalance loss functions while guaranteeing that the differences within the same group are insignificant while the differences between groups are significant. As shown in Fig. 3, we obtain the performance results of various imbalance loss functions on each dataset. We then use the Scott-Knott ESD test to analyze these results further. This test is used to derive the ranking of each model across different datasets. For instance, loss functions L_1 and L_3 are classified as the top-ranked group, while loss functions L_2 and L_5 are divided as the second-ranked group. There is no significant difference between the loss functions within either the top-ranked or the second-ranked group. However, according to the Scott-Knott ESD test, L_1 and L_3 in the top-ranked group are significantly better than L_2 and L_5 in the second-ranked group.

5. Experimental results

5.1. RQ1: Do imbalance loss functions improve the performance of DLVD models?

Approach: To address RQ1, we first evaluate the impact of employing nine widely used imbalance loss functions and CE loss on two DLVD models (LineVul and ReVeal) across four datasets (Devign, Reveal, Bigvul, and Juliet) in terms of six evaluation metrics (Precision, Recall, F1-score, MCC, AUC, and H-measure).

Visualization: Fig. 4(a)–(f) (for LineVul model) and Fig. 5(a)–(f) (for ReVeal model) depict the average performance of the loss functions on four vulnerability datasets in Precision, Recall, F1-score, MCC, AUC and H-measure metrics. Darker cell colors indicate better average performance. Additionally, employing the Scott-Knott ESD statistical test, Fig. 6(a)–(f) (for LineVul model) and Fig. 7(a)–(f) (for ReVeal model) present the ranking results of the loss functions based on these evaluation metrics. Lower rankings in cells indicate superior performance, with different rankings denoting significant differences between loss functions, while identical rankings suggest no significant difference.

Results:

(1) *Performance comparison: LineVul vs. ReVeal.*

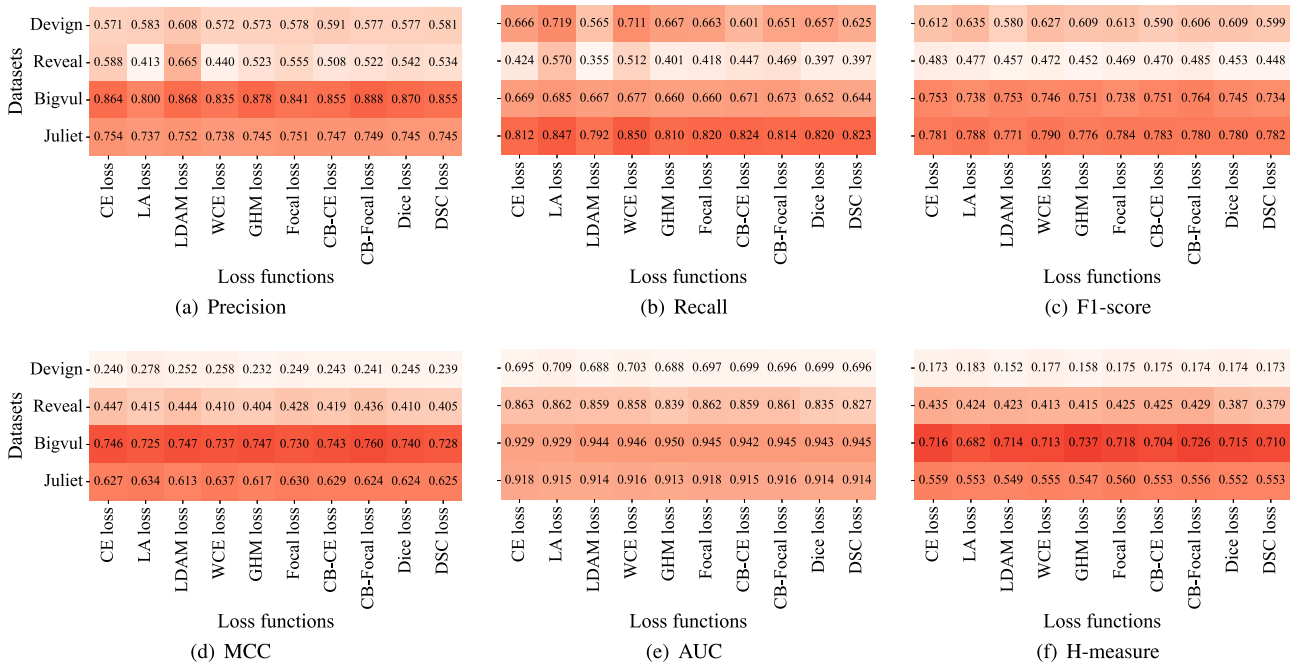


Fig. 4. The average value of loss functions for all datasets based on LineVul model.

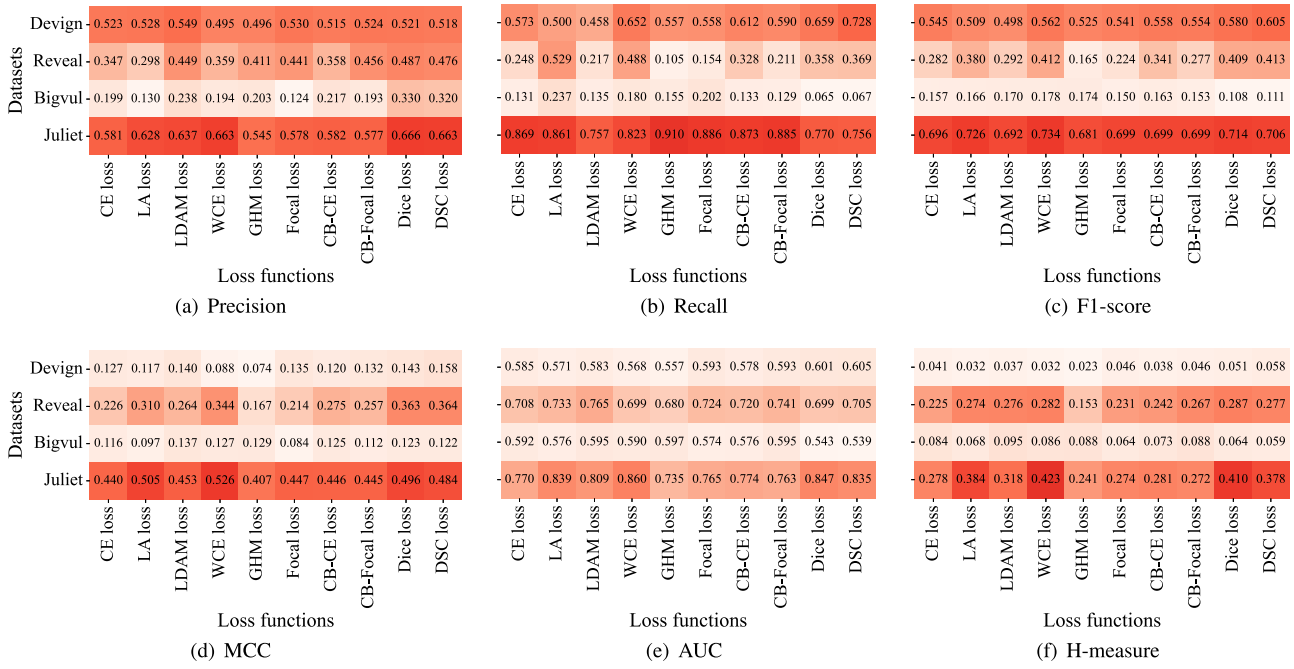


Fig. 5. The average value of loss functions for all datasets based on ReVeal model.

As can be seen from Figs. 4–5, the average performance of the LineVul model is better than that of the ReVeal model on all metrics. On the one hand, on the two extremely imbalanced datasets (i.e., Reveal and Bigvul), the LineVul model achieves a significantly greater performance improvement over the ReVeal model compared to the two nearly balanced datasets (i.e., Devign and Juliet). Specifically, on the Reveal dataset, LineVul surpasses ReVeal by 0.121 in Precision, 0.138 in Recall, 0.147 in F1-score, 0.143 in MCC, 0.135 in AUC, and 0.164 in H-measure. Similarly, on the Bigvul dataset, the average performance difference between LineVul and ReVeal is significant, with LineVul leading by 0.641 in Precision, 0.523 in Recall, 0.594 in F1-score, 0.623 in MCC, 0.364 in AUC, and 0.637 in H-measure. On the other hand, on the nearly balanced datasets, the performance difference is smaller: LineVul leads by 0.061 on the Devign dataset and 0.077 on the Juliet dataset in the F1-score. It is worth mentioning that on the Bigvul dataset, the average values of all metrics except H-measure for the LineVul model are above 0.640, whereas the average values for the ReVeal model range from 0.064 to 0.597 across all metrics except H-measure.

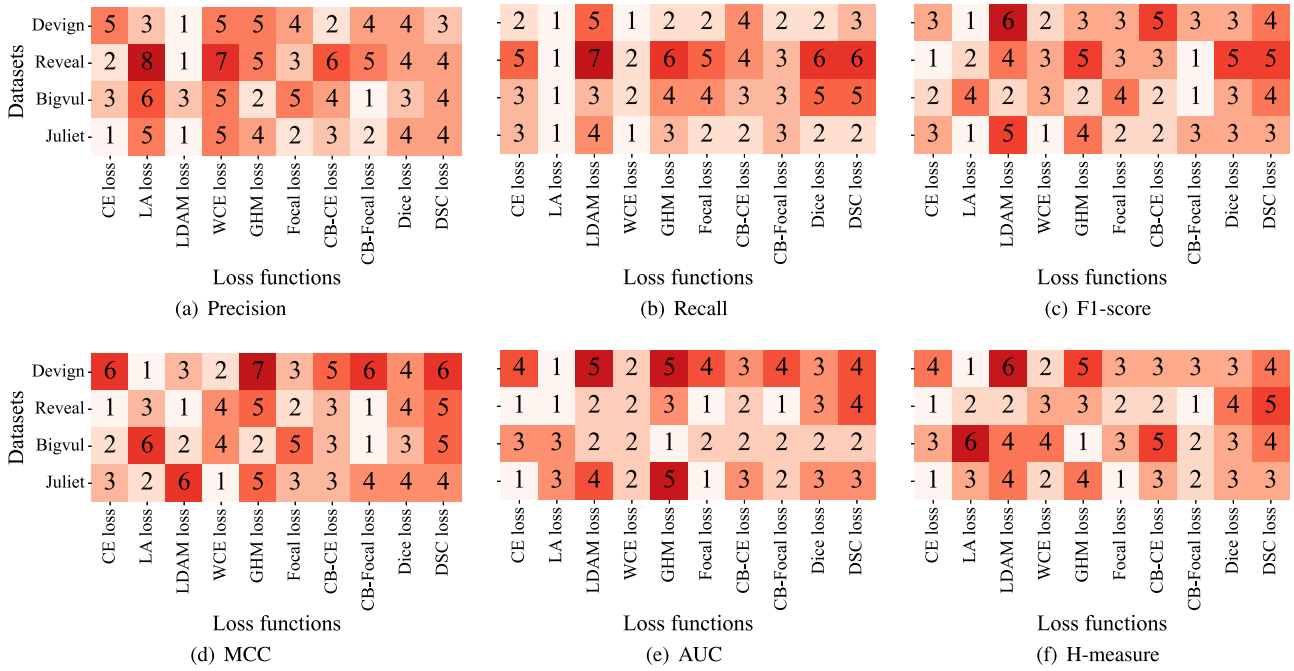


Fig. 6. The Scott-Knott ESD ranking of loss functions for all datasets based on LineVul model.

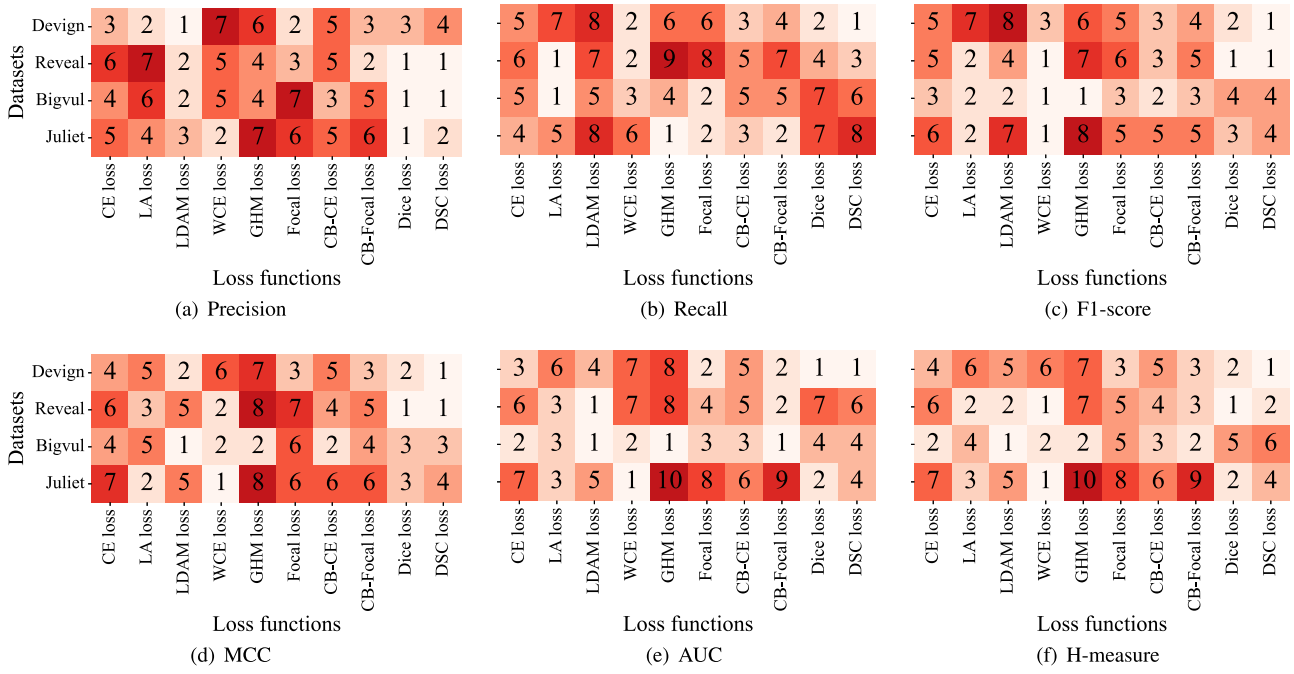


Fig. 7. The Scott-Knott ESD ranking of loss functions for all datasets based on ReVeal model.

> **Finding 1:** LineVul outperforms ReVeal across all metrics on all four datasets, with significantly greater advantages on extremely imbalanced datasets compared to nearly balanced ones.

(2) *Model performance on Precision metric.*

LineVul: As shown in Fig. 4(a), in the Precision metric, LDAM loss on LineVul model demonstrates the best average performance across two datasets (i.e., Devign and Reveal). Specifically, LDAM loss achieves 0.608 on Devign and 0.665 on the Reveal dataset. These findings show that LDAM loss improves by 6.5 % and 13.1 % compared to CE loss in Devign and Reveal, respectively. On the extremely imbalanced dataset (Bigvul), CB-Focal loss achieves the best score, with a value of 0.888. This value represents an improvement of 2.8 % compared to CE loss. On the Juliet dataset, CE loss and LDAM loss rank in the top two, respectively, and their performance is equivalent, with a difference of only 0.002 and a gap percentage of 0.2 %. Analyzing their Scott-Knott ESD ranking results shown in Fig. 6(a), we obtain similar conclusions as the average performance results. LDAM loss demonstrates the highest Scott-Knott ESD ranking for the Precision metric across three datasets (i.e., Devign, Reveal, and Juliet). On the

extremely imbalanced dataset (Bigvul), CB-Focal loss ranks in the top one. On the Juliet dataset, CE loss and LDAM loss both rank in the top one. Specially, in all imbalance loss functions, only LDAM loss is not lower than CE loss in the Scott-Knott ESD rankings for all four datasets.

ReVeal: Fig. 5(a) illustrates that on the Devign dataset, LDAM loss exhibits the best score, with a value of 0.549. This value represents an improvement of 5.0 % compared to CE loss. On the two extremely imbalanced datasets (Reveal and Bigvul), Dice loss achieves the best scores, with the values of 0.487 and 0.330, respectively. These results show a significant improvement of 40.3 % for the Reveal dataset and 65.8 % for the Bigvul dataset, compared to the performance of CE loss. On the Juliet dataset, Dice loss also achieves the highest Precision value of 0.666, marking a 14.6 % improvement over CE loss. Moreover, we find the Scott-Knott ESD ranking in Fig. 7(a): LDAM loss has the highest Scott-Knott ESD ranking of the Devign dataset. On the two extremely imbalanced datasets (Reveal and Bigvul), both Dice loss and DSC loss rank the best. On the Juliet dataset, Dice loss also exhibits the best ranking performance. In all imbalance loss functions, only LDAM loss and Dice loss are not lower than CE loss in the Scott-Knott ESD rankings for all four datasets.

> **Finding 2:** Regarding the Precision metric, only LDAM loss consistently outperforms CE loss across all four datasets and in both the LineVul and ReVeal models. LDAM loss with LineVul achieves the best Precision performance across all datasets.

(3) *Model performance on Recall metric.*

LineVul: Fig. 4(b) shows that LA loss has the best performance of the three datasets (i.e., Devign, Reveal, and Bigvul) on the Recall metric. LA loss achieves an average Recall value of 0.719 on the Devign dataset, 0.570 on the Reveal dataset, and 0.685 on the Bigvul dataset. Compared with CE loss, LA loss shows improvements of 2.4 %–34.4 % across the three datasets. On the Juliet dataset, LA loss and WCE loss rank in the top two, respectively, and their performance is equivalent, with a difference of only 0.003 and a gap percentage of 0.3 %. WCE loss and LA loss achieve Recall scores of 0.850 and 0.847, respectively, which are 4.7 % and 4.3 % higher than CE loss. Furthermore, the Scott-Knott ESD rankings, presented in Fig. 6(b), show that LA loss achieves the top one across all four datasets. WCE loss achieves first place in the Scott-Knott ESD rankings on both the Devign dataset and the Juliet dataset. In particular, LA loss, WCE loss, and CB-Focal loss outperform CE loss in the Scott-Knott ESD rankings on all four datasets.

ReVeal: By analyzing the average values presented in Fig. 5(b), the following patterns emerge: On the Devign dataset, DSC loss attains the top Recall value of 0.728, reflecting a 27.1 % improvement over CE loss. On the two extremely imbalanced datasets (Reveal and Bigvul), LA loss performs best and achieves the best scores, with the Recall scores of 0.529 and 0.237, respectively. These results show a significant improvement of 113.3 % for the Reveal dataset and 80.9 % for the Bigvul dataset, compared to the performance of CE loss. On the Juliet dataset, GHM loss exhibits the best Recall score, with a value of 0.910. The improvement of GHM loss compared to CE loss is 4.7 %. Fig. 7(b) indicates that DSC loss achieves the best ranking on the Devign dataset. On the two extremely imbalanced datasets (Reveal and Bigvul), LA loss achieves the highest Scott-Knott ESD ranking. GHM loss ranks first in the Scott-Knott ESD rankings on the Juliet dataset. In particular, only CB-CE loss is better than CE loss in the Scott-Knott ESD rankings for all four datasets.

> **Finding 3:** In terms of Recall, LA loss performs best across all four datasets on LineVul model. Additionally, the ReVeal model trained with LA loss achieves the best performance on two highly imbalanced datasets.

(4) *Model performance on F1 metric.*

LineVul: Analyzing the average values presented in Fig. 4(c), LA loss achieves the highest F1-score value of 0.635 on the Devign dataset, marking a 3.8 % improvement over CE loss. On the two extremely imbalanced datasets (Reveal and Bigvul), CB-Focal loss obtains the best scores, with values of 0.485 and 0.764, respectively. These results represent an improvement of 0.4 % (Reveal) and 1.4 % (Bigvul) compared to CE loss. On the Juliet dataset, WCE loss attains the best F1-score value of 0.790, demonstrating a 1.2 % improvement over CE loss. Additionally, the Scott-Knott ESD rankings in Fig. 6(c) reveal that on the Devign dataset, LA loss outperforms other loss functions and ranks the top one. On the two extremely imbalanced datasets (Reveal and Bigvul), CB-Focal loss exhibits the best rankings. Moreover, on the Reveal dataset, CE loss also achieves the top ranking. On the Juliet dataset, LA loss and WCE loss both rank first in the Scott-Knott ESD rankings. In particular, in all imbalance loss functions, only CB-Focal loss is not lower than CE loss in the Scott-Knott ESD rankings for all four datasets.

ReVeal: Regarding the F1-score comprehensive metric, Fig. 5(c) illustrates that DSC loss exhibits the optimal value of 0.605 on the Devign dataset, demonstrating an 11.0 % improvement over CE loss. Dice loss also attains the top F1-score value of 0.413 on the Reveal dataset. This value represents substantial improvements of 46.5 % compared to CE loss. On the Bigvul and Juliet dataset, WCE loss performs best and achieves the F1-score of 0.178 and 0.734, respectively. These results reveal the improvement of 13.4 % for the Bigvul dataset and 5.5 % for the Juliet dataset, compared to the performance of CE loss. Based on the Scott-Knott ESD statistical test shown in Fig. 7(c), on the Devign dataset, DSC loss is ranked as the top one. On the Reveal dataset, WCE loss, Dice loss, and DSC loss all rank first in the Scott-Knott ESD rankings. On the Bigvul dataset, both WCE loss and GHM loss exhibit the best rankings. WCE loss has the highest Scott-Knott ESD ranking on the Juliet dataset. It is worth noting that WCE loss, CB-CE loss, and CB-Focal loss outperform CE loss in the Scott-Knott ESD ranking across all four datasets. The Scott-Knott ESD ranking results are consistent with the patterns observed in the average value results.

> **Finding 4:** On the comprehensive F1-score metric, only CB-Focal loss consistently outperforms CE loss across all four datasets and in both LineVul and ReVeal models.

(5) *Model performance on MCC metric.*

LineVul: The experimental results reflected by the MCC metric and the F1-score metric show similar patterns. The average MCC values in Fig. 4(d) reveal that on the Devign dataset, LA loss exhibits the best MCC score of 0.278. Compared to CE loss, LA loss improves by 15.8 %. On the Reveal dataset, CE loss achieves the best MCC value of 0.447. On the extremely imbalanced dataset (Bigvul), CB-Focal loss obtains the best average performance, with a value of 0.760. The improvement of CB-Focal loss compared to CE loss is 1.9 %. On the Juliet dataset, WCE loss achieves the highest MCC value of 0.637, representing an improvement of 1.6 % compared to CE loss. The Scott-Knott ESD rankings depicted

in Fig. 6(d) further support this trend: On the Devign dataset, LA loss exhibits the highest Scott-Knott ESD ranking. On the two extremely imbalanced datasets (Reveal and Bigvul), CB-Focal loss achieves the top ranking. Moreover, CE loss and LDAM loss also obtain the top ranking on the Reveal dataset. WCE loss ranks highest and first among all evaluated loss functions on the Juliet dataset, indicating its superior performance.

ReVeal: The experimental results reflected by the MCC metric and the F1-score metric show consistent conclusions. In terms of MCC, Fig. 5(d) indicates that DSC loss exhibits the best MCC values of 0.158 and 0.364 on the Devign and Reveal datasets, respectively. These results represent improvements of 24.4 % on the Devign dataset and 61.1 % on the Reveal dataset compared to CE loss. On the Bigvul dataset, LDAM loss obtains the best score, with the value of 0.137. This value represents an improvement of 18.1 % compared to CE loss. On the Juliet dataset, WCE loss achieves the average MCC value of 0.526, demonstrating a 19.5 % improvement over CE loss. Observing the Scott-Knott ESD ranking results shown in Fig. 5(d), we find that DSC loss exhibits the best ranking performance on the Devign dataset. On the Reveal dataset, Dice loss and DSC loss rank in the first place. On the Bigvul dataset, LDAM loss has the highest Scott-Knott ESD ranking. On the Juliet dataset, WCE loss achieves the best ranking. Based on the Scott-Knott ESD rankings across the four datasets, both Dice loss and DSC loss outperform CE loss.

> **Finding 5:** The performance rankings of the LineVul model and the ReVeal model on their respective MCC metric are similar to those shown by the F1-score metric.
>
> **Finding 6:** For comprehensive performance (i.e., high F1-score and MCC), CB-Focal loss achieves the best results on extremely imbalanced datasets, and LA loss performs best on nearly balanced datasets on the LineVul model.

(6) *Model performance on AUC metric.*

LineVul: Analyzing the average values presented in Fig. 4(e), LA loss achieves the highest AUC value of 0.709 on the Devign dataset, marking a 2.0 % improvement over CE loss. On the Reveal dataset, CE loss shows the best AUC performance. However, the LA loss follows closely, with a difference of only 0.001. On the Bigvul dataset, GHM loss exhibits the optimal value of 0.950. Moreover, all imbalance loss functions is better than CE loss, with the improvement of 0–2.3 %. On the Juliet dataset, CE loss and Focal loss both achieve the optimal value of 0.918. Additionally, the Scott-Knott ESD rankings in Fig. 6(e) reveal that on the Devign dataset, LA loss outperforms other loss functions and ranks the top one. On the Reveal dataset, CE loss, LA loss, Focal loss, and CB-Focal loss exhibit the best rankings. On the Bigvul dataset, GHM loss ranks first in the Scott-Knott ESD rankings. Additionally, CE loss is less effective on the extremely imbalanced Bigvul dataset. On the Juliet dataset, CE loss and Focal loss both exhibit the best rankings.

ReVeal: Regarding the AUC metric, Fig. 5(e) illustrates that DSC loss exhibits the optimal value of 0.605 on the Devign dataset, demonstrating an 3.4 % improvement over CE loss. LDAM loss also attains the top AUC value of 0.765 on the Reveal dataset. This value represents substantial improvements of 8.1 % compared to CE loss. On the Bigvul dataset, GHM loss performs best and achieves the AUC of 0.597. This result reveals the improvement of 0.1 % compared to the performance of CE loss. On the Juliet dataset, WCE loss achieves the best value of 0.860, demonstrating a 11.7 % improvement over CE loss. Based on the Scott-Knott ESD statistical test shown in Fig. 7(e), on the Devign dataset, Dice loss and DSC loss are both ranked as the top one. On the Reveal dataset, LDAM loss ranks first in the Scott-Knott ESD rankings. On the Bigvul dataset, LDAM loss, GHM loss, and CB-Focal loss exhibit the best rankings. WCE loss has the highest Scott-Knott ESD ranking on the Juliet dataset. The Scott-Knott ESD ranking results are consistent with the patterns observed in the average value results.

(7) *Model performance on H-measure metric.*

LineVul: The average H-measure values in Fig. 4(f) reveal that on the Devign dataset, LA loss exhibits the best H-measure score of 0.183. Compared to CE loss, LA loss improves by 5.8 %. On the Reveal dataset, CE loss achieves the best H-measure value of 0.435. However, the CB-Focal loss follows closely, with a difference of only 0.006. On the extremely imbalanced dataset (Bigvul), GHM loss obtains the best average performance, with a value of 0.737. The improvement of GHM loss compared to CE loss is 2.9 %. On the Juliet dataset, Focal loss achieves the highest H-measure value of 0.560, representing an improvement of 0.1 % compared to CE loss. The Scott-Knott ESD rankings depicted in Fig. 6(f) further support this trend: On the Devign dataset, LA loss exhibits the highest Scott-Knott ESD ranking. On the Reveal dataset, both CE loss and CB-Focal loss achieve the top ranking. On the Bigvul dataset, GHM loss has the highest Scott-Knott ESD ranking. CE loss and Focal loss obtain the top ranking on the Juliet dataset.

ReVeal: In terms of H-measure, Fig. 5(f) indicates that DSC loss exhibits the best H-measure values of 0.058 on the Devign dataset. The improvement of DSC loss compared to CE loss is 41.5 %. On the Reveal dataset, Dice loss obtains the best score, with the value of 0.287, demonstrating a 27.6 % improvement over CE loss. On the Bigvul dataset, LDAM loss obtains the best score, with the value of 0.095. This value represents an improvement of 13.1 % compared to CE loss. On the Juliet dataset, WCE loss achieves the highest H-measure value of 0.423, representing an improvement of 52.2 % compared to CE loss. Observing the Scott-Knott ESD ranking results shown in Fig. 7(f), we find that DSC loss exhibits the best ranking performance on the Devign dataset. On the Reveal dataset, WCE loss and Dice loss rank in the first place. On the Bigvul dataset, LDAM loss has the highest Scott-Knott ESD ranking. On the Juliet dataset, WCE loss achieves the best ranking.

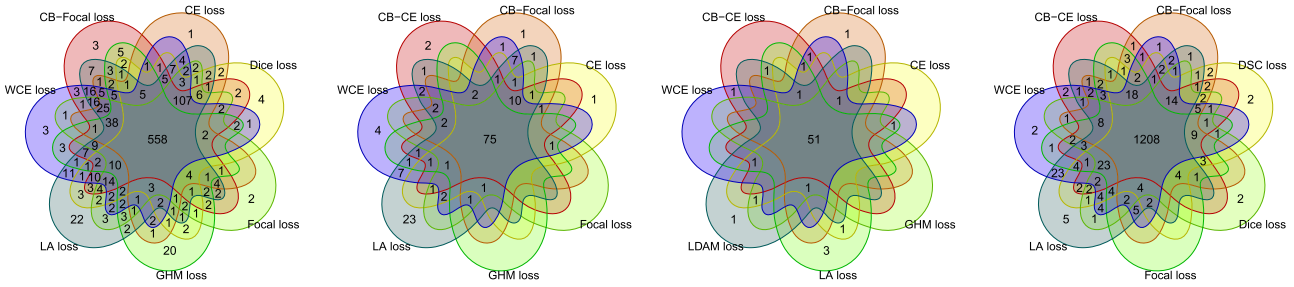
> **Finding 7:** Considering the results of all six metrics, CE loss is not the optimal choice when facing vulnerability datasets with class imbalance, because there are always several imbalance loss functions that perform better than CE loss in specific situations.

5.2. RQ2: What degree of orthogonality is observed among the studied loss functions in detecting vulnerable code snippets?

RQ1 investigates the performance of various loss functions across four datasets using six evaluation metrics. Since the primary objective of DLVD is to effectively detect vulnerable code snippets, this research question focuses on evaluating the performance of different loss functions in detecting unique vulnerable code snippets within each dataset (i.e., vulnerable code snippets that are correctly identified by one loss function but misclassified by all other loss functions).

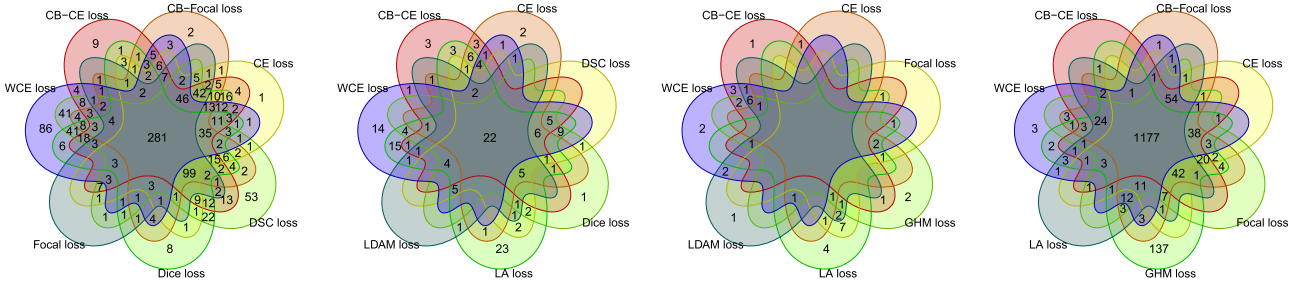
Approach: We employ orthogonality analysis to evaluate the performance of different loss functions in detecting unique vulnerable code snippets. These vulnerable code snippets are correctly classified by one loss function but misclassified by all others. Based on the Recall rankings detailed in Section 5.1, we select the top seven performing loss functions from each dataset for this analysis.

Visualization: Figs. 8 and 9 depict the results of the orthogonality analysis for these seven loss functions, focusing on correctly predicting vulnerable code snippets. In these two figures, the values within the overlapping regions indicate the number of vulnerable code snip-



(a) Devign (b) Reveal (c) Bigvul (d) Juliet

Fig. 8. The orthogonality results of these loss functions results for vulnerable code snippets on LineVul model.



(a) Devign (b) Reveal (c) Bigvul (d) Juliet

Fig. 9. The orthogonality results of these loss functions results for vulnerable code snippets on ReVeal model.

pets correctly identified by the respective loss functions. The individual area for each loss function corresponds to the count of vulnerable code snippets accurately identified solely by that loss function. This count reflects the orthogonality of each loss function. A higher outermost value for a loss function indicates greater orthogonality, signifying its superior capability in detecting vulnerable code snippets. For instance, in Fig. 8(a), the value 558 denotes that all seven loss functions collectively identified 558 vulnerable code snippets accurately. Specifically, LA loss independently identified 22 vulnerable code snippets, representing an orthogonality of 22.

Result: When utilizing LineVul as the DLVD model, LA loss exhibits the highest orthogonality score of 22 on the Devign dataset, as shown in Fig. 8(a). Moreover, GHM loss achieves the second highest orthogonality of 20. They considerably surpasses CE loss, which has a score of 1. Additionally, LA loss outperforms the other top-ranked loss function, WCE loss, which achieves a score of 3 on the Devign dataset. On the Reveal dataset, LA loss exhibits the best orthogonality of 23, as depicted in Fig. 8(b). LA loss outperforms the second-ranked method WCE loss by a margin of 19 orthogonality score. Similarly, on the Bigvul dataset, LA loss achieves the highest orthogonality of 3, as shown in Fig. 8(c). In contrast, CB-CE loss, WCE loss, GHM loss, and CE loss all obtain an orthogonality score of 0. Lastly, the orthogonality analysis presented in Fig. 8(d) reveals that LA loss attains the top orthogonality score of 5 on the Juliet dataset, outperforming WCE loss, the same first-ranked loss function, which achieves a score of 2.

When utilizing ReVeal as the DLVD model: On the Devign dataset Fig. 9(a), WCE loss exhibits the highest orthogonality of 86, significantly outperforming CE loss, which attains the orthogonality score of 1. Meanwhile, DSC loss achieves the second highest orthogonality of 53. LA loss demonstrates the best orthogonality of 23 on the Reveal dataset Fig. 9(b). LA loss achieves the highest orthogonality of 4 on the Bigvul dataset Fig. 9(c), significantly outperforming CE loss, which can not identify any unique vulnerable code snippets. Lastly, GHM loss attains the top orthogonality of 137 on the Juliet dataset Fig. 9(d), notably surpassing WCE loss. In contrast, CE loss, LA loss, Focal loss, CB-CE loss, and CB-Focal loss all attain an orthogonality score of 0.

> **Finding 8:** LA loss shows the highest orthogonality on LineVul model across all datasets, indicating its effectiveness in improving DLVD models for detecting more unique vulnerable code snippets. Additionally, other loss functions (e.g., WCE loss, GHM loss, and DSC loss) with high Recall also perform well in orthogonality.

5.3. RQ3: Why do imbalance loss functions alleviate the class imbalance problem?

Approach: In RQ1, we study the impact of various imbalance loss functions on the performance of DLVD models. From the experimental results, we find that the overall performance of the LineVul model is better than that of the ReVeal model. To analyze the interpretability of the DLVD model trained with the imbalance loss functions, we apply the LIT (Tenney et al., 2020) to the LineVul model, which is widely used in software engineering (Steenhoek, Rahman, Jiles, & Le, 2023). LIT can identify the most important tokens in a code snippet that contribute to the predictions of DLVD model. Specifically, it uses the integrated gradient method to calculate the gradient and contribution of each token. LIT then combines these gradients and contributions into a score to measure the influence of each token on the model prediction. Higher scores indicate tokens that have the greater impact on the model's decision-making process. Following the reasearch setting (Steenhoek et al., 2023), we calculate the score for each line of code snippet by summing the scores of all tokens within that line. We consider that the lines of code with higher scores contribute more to the DLVD model's decision making. The analysis details are as follows: (1) We exclude the score of the first line, which contains the function header. Function headers typically include multiple parameter inputs and function names, resulting in a sig-

nificantly higher token count and score compared to other code lines. Furthermore, vulnerabilities in functions usually arise in the lines of code within the function itself. (2) We identify the top four scoring lines of code, which are considered to contribute more to vulnerabilities and conduct an in-depth analysis of these lines.

Based on the performance in RQ1, we select the imbalance loss functions that exhibit the best performance in terms of F1-score and MCC metrics. Since real-world vulnerability datasets are usually extremely imbalanced, we select an example of interpretability analysis on two extremely imbalanced datasets (i.e., Reveal and Bigvul) in our study. Specifically, we choose CB-Focal loss for the Reveal and Bigvul datasets. We then compare their vulnerability detection performance with that of CE loss. To obtain generalizable results, we perform stratified sampling of vulnerable code snippets from each dataset based on code length distribution: short (<10 lines), medium (10-100 lines), and long (>100 lines). From each dataset, we select 20 vulnerable code snippets (accounting for 0.89 % for the Reveal dataset and 2.1 % for the Bigvul dataset) according to this length distribution to ensure that our analysis captures a diverse range of code complexities and structures. This approach allows us to better understand how different loss functions perform across various code lengths, which is crucial for drawing meaningful conclusions. We then apply LIT to these vulnerable code snippets to elucidate the performance differences between CB-Focal loss and CE loss. To illustrate this, we choose a representative vulnerable code snippet from each high imbalanced dataset, as demonstrated in Figs. 10–11, respectively.

Visualization: Figs. 10–11 present the interpretability analysis of vulnerable code snippets for DLVD models trained with imbalance loss functions across two extremely imbalanced datasets. These two representative vulnerable code snippets are incorrectly predicted under CE loss but correctly predicted when using CB-Focal loss. In these figures, the red-shaded code line has the highest score, while the yellow-shaded code line has a score between the Top 2 and Top 4. The red-colored code line indicates the source of vulnerability.

Result: In the Reveal dataset, we select 3 short, 14 medium, and 3 long code snippets based on their distribution (17 %, 67.23 %, and 15.67 %, respectively). For the short code snippets, all three samples are correctly predicted by both the CE loss and CB-Focal loss. Among the 14 medium-length code snippets, six are accurately predicted by both loss functions, while the remaining eight are misclassified under CE loss but correctly identified when using CB-Focal loss. Notably, for these eight code snippets, the model trained with CB-Focal loss assigns Top 5 scores to the vulnerable lines in six cases. Regarding the three long code snippets, two are correctly predicted by both loss functions, while one is accurately identified only by the model trained with CB-Focal loss. Overall, among the 20 selected vulnerable code snippets, nine are correctly predicted by the model using CB-Focal loss but misclassified by the model using CE loss. For these nine snippets, the model trained with CB-Focal loss consistently assigns Top 5 scores to the lines identified as vulnerable.

In the Bigvul dataset, we select 4 short, 13 medium, and 3 long code snippets based on their natural distribution (19.18 %, 64.22 %, and 16.60 %, respectively). For the short code snippets, all four are correctly predicted by the CB-Focal loss, while the CE loss misclassifies all of them. The vulnerable lines in these snippets are consistently ranked within the Top 2 by the model trained with CB-Focal loss. Among the 13 medium-length code snippets, four are accurately predicted by both CB-Focal loss and CE loss, while the model trained with CB-Focal loss correctly identifies nine additional snippets that are misclassified by CE loss. Notably, in all nine of these cases, the model using CB-Focal loss consistently assigned Top 5 scores to the lines identified as vulnerable. For the long code snippets, two are correctly predicted by both loss functions, while one is accurately identified only by the model trained with CB-Focal loss. Overall, among the 14 code snippets where CB-Focal loss outperforms CE loss, the model trained with CB-Focal loss consistently assigns Top 5 scores to the vulnerable lines in 13 of these snippets. These findings

Line	Score	Vulnerable Code Snippet (**Reveal**)
1	~~0.246855~~	static int com_connect(String *buffer, char *line) {
2	0.339551	char *tmp, buff[256];
3	0.015939	my_bool save_rehash = opt_rehash;
4	0.002487	int error;
5	0.036139	bzero(buff, sizeof(buff));
6	0.014313	if (buffer) {
7	0.058071	tmp = strmake(buff, line, sizeof(buff) - 2);
8	0.017809	#ifdef EXTRA_DEBUG tmp[1] = 0;
9	0.011232	#endif tmp = get_arg(buff, 0);
10	0.007812	if (tmp && *tmp) {
11	0.004527	my_free(current_db);
12	0.004491	current_db = my_strdup(tmp, MYF(MY_WME));
13	0.004395	tmp = get_arg(buff, 1);
14	0.000593	if (tmp) {
15	0.002204	my_free(current_host);
16	0.002999	current_host = my_strdup(tmp, ↪ MYF(MY_WME));
17	0.000131	}
18	0.000154	}
19	0.000558	else {
20	0.002170	opt_rehash = 0;
21	0.001163	}
22	0.070822	buffer->length(0);
23	0.000373	}
24	0.001250	else ↪ opt_rehash = 0;
25	0.008794	error = sql_connect(current_host, current_db, ↪ current_user, opt_password, 0);
26	0.001235	opt_rehash = save_rehash;
27	0.001605	if (connected) {
28	0.013421	sprintf(buff, "Connection id: %lu", ↪ mysql_thread_id(&mysql));
29	0.002628	put_info(buff, INFO_INFO);
30	0.006199	sprintf(buff, "Current database: %.128s\n", ↪ current_db ? current_db : "*** NONE ***");
31	0.002183	put_info(buff, INFO_INFO);
32	0.000398	}
33	0.000874	return error;
34	0.001933	}

Fig. 10. The representative vulnerable code snippet from the Reveal dataset.

further illustrate that imbalanced loss functions enhance the learning process from vulnerable code snippets, allowing the model to capture more distinguishing features and improving its ability to identify vulnerable lines of code.

Fig. 10 presents a vulnerable code snippet from the Reveal dataset exhibiting a *Buffer Overflow* vulnerability, where the lack of validation for the size of the input `line` leads to potential memory overflow. While the LineVul model trained with CE loss incorrectly classifies the code snippet as non-vulnerable, it is accurately identified as vulnerable by the model trained with CB-Focal loss. Specifically, the vulnerability arises from the use of the `strmake` function on line 7, where the `buff` array, which is defined with a size of 256, is passed as the destination buffer. The function call `strmake(buff, line, sizeof(buff) - 2)` aims to copy the contents of `line` into `buff`, but it does not properly account for the possibility that `line` might exceed the allocated size for `buff`. When the length of `line` exceeds 254 characters (i.e., when `line` is greater than 254), this results in a buffer overflow as `strmake` will attempt to write more data into `buff` than it can hold. This could overwrite adjacent memory, leading to undefined behavior, such as data corruption, crashes, or even potential security vulnerabilities like arbitrary code execution or information leakage.

As shown in the Fig. 10, line 7 is the code line where the vulnerability exists in this function. Through LIT's analysis, we find that the code lines ranking Top 1 to Top 4 contain the vulnerable code line (line 7). In addition, the score of line 7 ranks in the Top 3. This demonstrates that the LineVul model trained with CB-Focal loss can effectively prioritize and focus on code lines containing vulnerabilities.

Fig. 11 presents a vulnerable code snippet from the Bigvul dataset exhibiting an *Integer Overflow* vulnerability. The LineVul model trained with CE loss fails to recognize the code snippet as vulnerable, whereas the model trained with CB-Focal loss correctly classifies it as vulnerable. The vulnerability arises in line 16, where the multiplication

Line	Score	Vulnerable Code Snippet (**Bigvul**)
1	~~0.118617~~	long long Chapters::Atom::GetTime(
		↪ const Chapters* pChapters, long long timecode)
2	0.004913	{
3	0.044016	if (pChapters == NULL)
4	0.043817	return -1;
5	0.083786	Segment* const pSegment = pChapters->m_pSegment;
6	0.055819	if (pSegment == NULL) // weird
7	0.043803	return -1;
8	0.075769	const SegmentInfo* const pInfo =
		↪ pSegment->GetInfo();
9	0.039825	if (pInfo == NULL)
10	0.043804	return -1;
11	0.076086	const long long timecode_scale =
		↪ pInfo->GetTimeCodeScale();
12	0.059811	if (timecode_scale < 1) // weird
13	0.043827	return -1;
14	0.039851	if (timecode < 0)
15	0.043809	return -1;
16	0.064348	const long long result = timecode_scale * timecode;
17	0.023934	return result;
18	0.004009	}

Fig. 11. The representative vulnerable code snippet from the Bigvul dataset.

operation involving `timecode_scale` and `timecode` can potentially result in a `long` overflow, leading to incorrect behavior. The code does not contain any checks or safeguards to prevent this overflow condition. Specifically, the function `Chapters::Atom::GetTime` receives a `timecode` parameter, which is multiplied by `timecode_scale` in line 16. If both `timecode_scale` and `timecode` are large enough, their product could exceed the maximum value representable by the `long` type (typically $2^{63} - 1$ for signed 64-bit integers). This overflow can cause the result to wrap around to a negative or erroneous value, producing unpredictable and potentially dangerous results.

As can be seen from Fig. 11, the vulnerability is located at line 16. Based on LIT's analysis, the vulnerable code line (line 16) is included in the Top 1 to 4 lines. This demonstrates the effectiveness of the LineVul model training with CB-Focal loss in focusing on vulnerable code lines.

> **Finding 9:** The imbalance loss functions adjust the loss weight or probability to direct the DLVD model's attention to code lines that contribute to vulnerabilities, thereby enhancing model performance.

6. Discussion

6.1. How do the hyperparameters of imbalance loss functions influence their effectiveness in mitigating class imbalance in DLVD models?

Given the potential for further optimization of the recommended imbalance loss functions to enhance DLVD model performance, we investigate the hyperparameter settings of the two best-performing loss functions (i.e., LA loss and CB-Focal loss) identified in Section 5.1. Specifically, we conduct a comprehensive exploration of key hyperparameters on two representative models (i.e., LineVul and ReVeal). Following the methodology of prior studies (Cui et al., 2019; Menon et al., 2021), we manually explore various hyperparameter values in the two loss functions, train the models, and determine the optimal settings based on performance. For LA loss, we examine the impact of the temperature scaling parameter τ, varying it within the set $\{0.0, 0.5, 1.0, 1.5, 2.0\}$, following the setting in prior work (Menon et al., 2021). For CB-Focal loss, we search over the combinations of the focusing parameter $\gamma \in \{0.5, 1.0, 2.0\}$ and the class-balancing parameter $\beta \in \{0.9, 0.99, 0.999, 0.9999\}$, as recommended by Cui et al. (2019). The results presented are based on a single representative run, as we observe minimal variance across five repeated experiments.

τ adjusts the strength of class prior influence on logits. Fig. 12(a) and (a) illustrate the performance of LA loss under varying τ values for the LineVul and ReVeal models, respectively. The experimental results show that for the LineVul model, LA loss achieves the best Precision and MCC when $\tau = 0.5$, while Recall and F1-score peak at $\tau = 1.5$. However, for AUC and H-measure, no single τ value consistently yields the best performance. Similarly, for the ReVeal model, Precision and MCC are highest at $\tau = 0.5$, Recall is best at $\tau = 2.0$, while no single τ value consistently yields the best F1-score, AUC, or H-measure across all parameter settings. These findings indicate that there is no single τ value that consistently yields the best performance across all evaluation metrics.

γ modulates the focus on hard-to-classify samples, whereas β is employed to calculate the effective number of samples, thereby accounting for class imbalance. Fig. 13(a) and (b) illustrate the performance of CB-Focal loss under varying γ and β values for the LineVul and ReVeal models, respectively. The experimental results reveal that for the LineVul model, CB-Focal loss achieves the best Precision with $\gamma = 2.0$ and $\beta = 0.99$, while both Recall and F1-score are highest when $\gamma = 1.0$ and $\beta = 0.9999$. CB-Focal loss achieves the highest AUC when $\gamma = 1.0$ and $\beta = 0.99$. However, there is no single (γ, β) combination that consistently performs best for MCC and H-measure. For the ReVeal model, CB-Focal loss attains the highest Precision with $\gamma = 1.0$ and $\beta = 0.9$, and the best MCC with $\gamma = 0.5$ and $\beta = 0.9999$. CB-Focal loss achieves the best H-measure with $\gamma = 2.0$ and $\beta = 0.999$. However, no consistent best combination is observed for Recall, F1-score, or AUC. These observations further confirm that no single combination of γ and β hyperparameters yields optimal performance across all evaluation metrics.

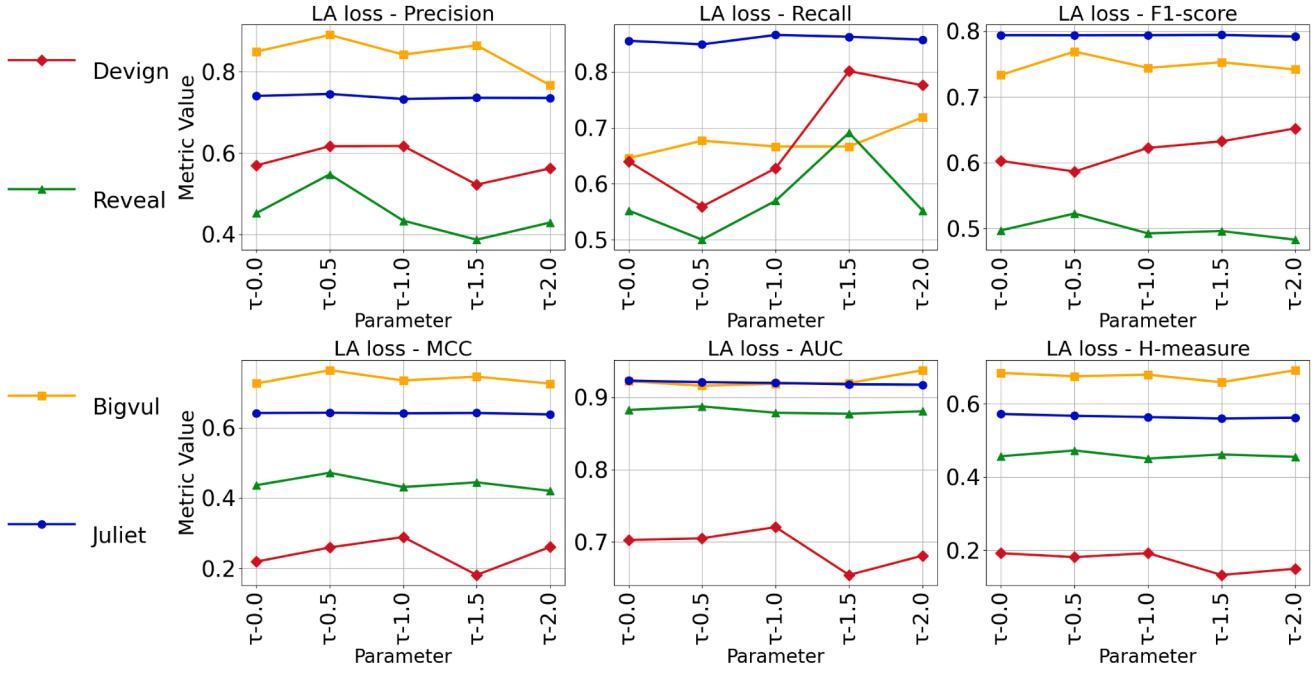
As detailed in Section 5.1, we adopt the default hyperparameter settings of LA loss and CB-Focal loss. Consequently, future research on specific datasets can be further studied based on our recommended hyperparameter settings.

6.2. Do imbalance loss functions effectively separate two classes?

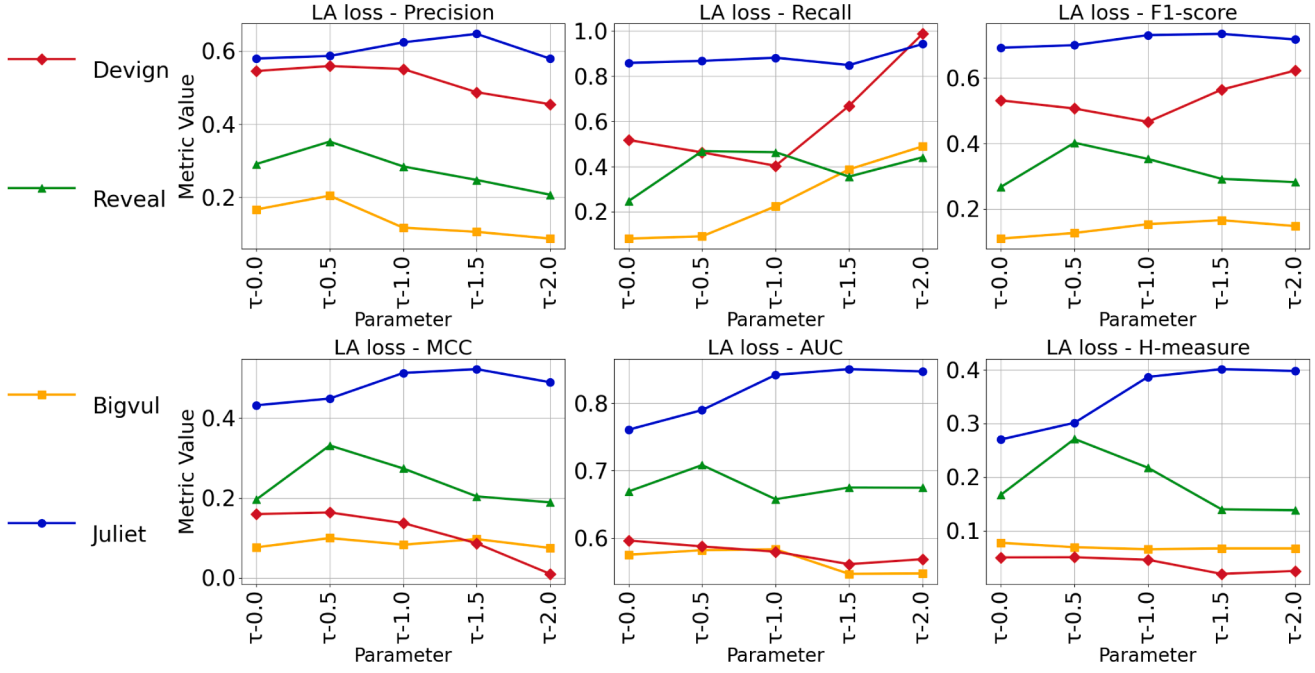
To intuitively understand the feature distributions of vulnerable and non-vulnerable code snippets, we conduct a feature space distribution analysis inspired by prior work (Chakraborty et al., 2022). Our goal is to assess whether the data is linearly separable and if the balance CE loss is sufficient for effective classification on imbalanced vulnerability datasets. We further investigate how different loss functions affect feature space separability by visualizing sample distributions from learned representations under both CE loss and imbalance loss functions.

Based on the findings of RQ1, we select the best-performing imbalance loss functions for both the LineVul and ReVeal models, evaluated on two highly imbalanced datasets (i.e., Reveal and Bigvul). Using the t-SNE technique (Van der Maaten & Hinton, 2008), we visualize the feature space separability, as shown in Fig. 14. The plot depicts vulnerable code snippets as red circles (∘) and non-vulnerable ones as blue plus signs (+). Each subplot compares the feature space of the best imbalance loss against CE loss for a specific model-dataset pair. For example, the label "LineVul-Reveal" in Fig. 14(a) denotes the comparison of the best imbalance loss function and CE loss on the Reveal dataset using the LineVul model. Specifically, Figs. 14(a) and (b) present the feature spaces learned by the LineVul model on the Reveal and Bigvul datasets, comparing the representations obtained using CE loss versus CB-Focal loss. Similarly, Fig. 14(c) and (d) illustrate the feature spaces generated by the ReVeal model on these datasets, contrasting CE loss with WCE loss.

In the feature space produced by CE loss, as shown Figs. 14(a) –14(d), a significant overlap is observed between vulnerable code snippets (represented by red circles) and non-vulnerable code snippets (represented by blue plus signs). This overlap hinders the model's ability to define clear decision boundaries between the two classes. However, this



(a) LineVul

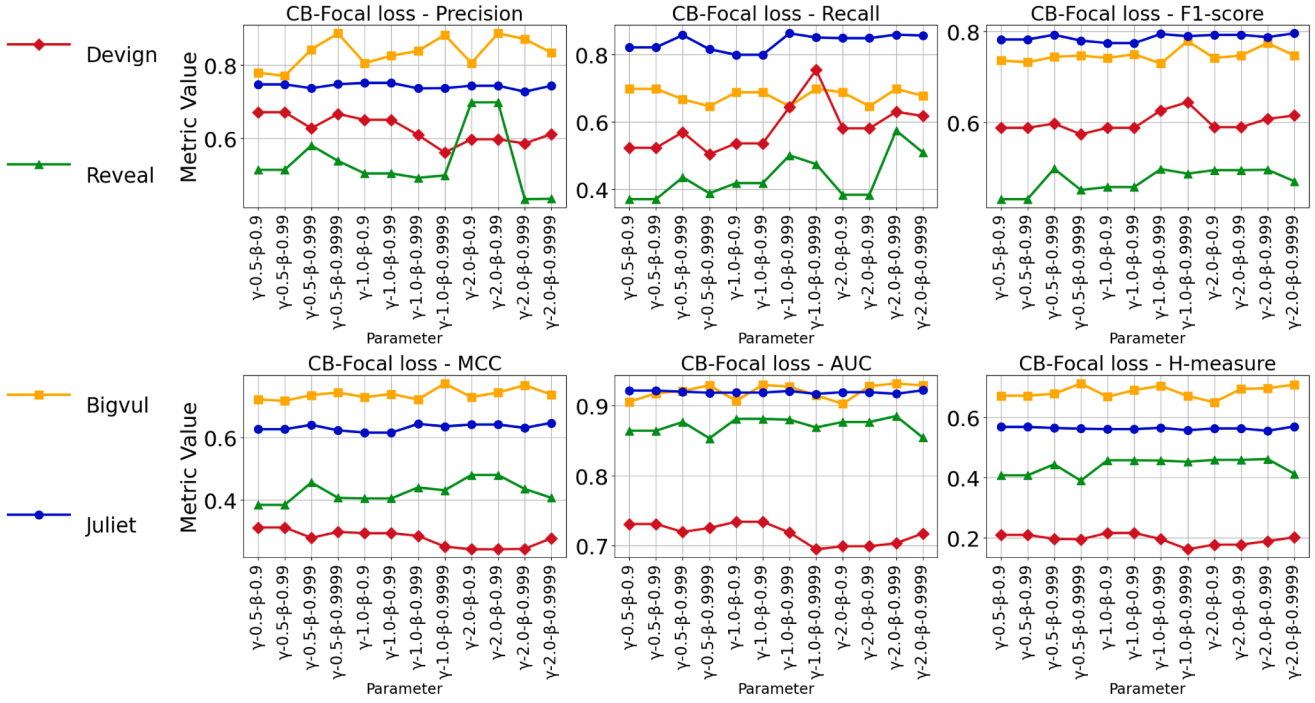


(b) ReVeal

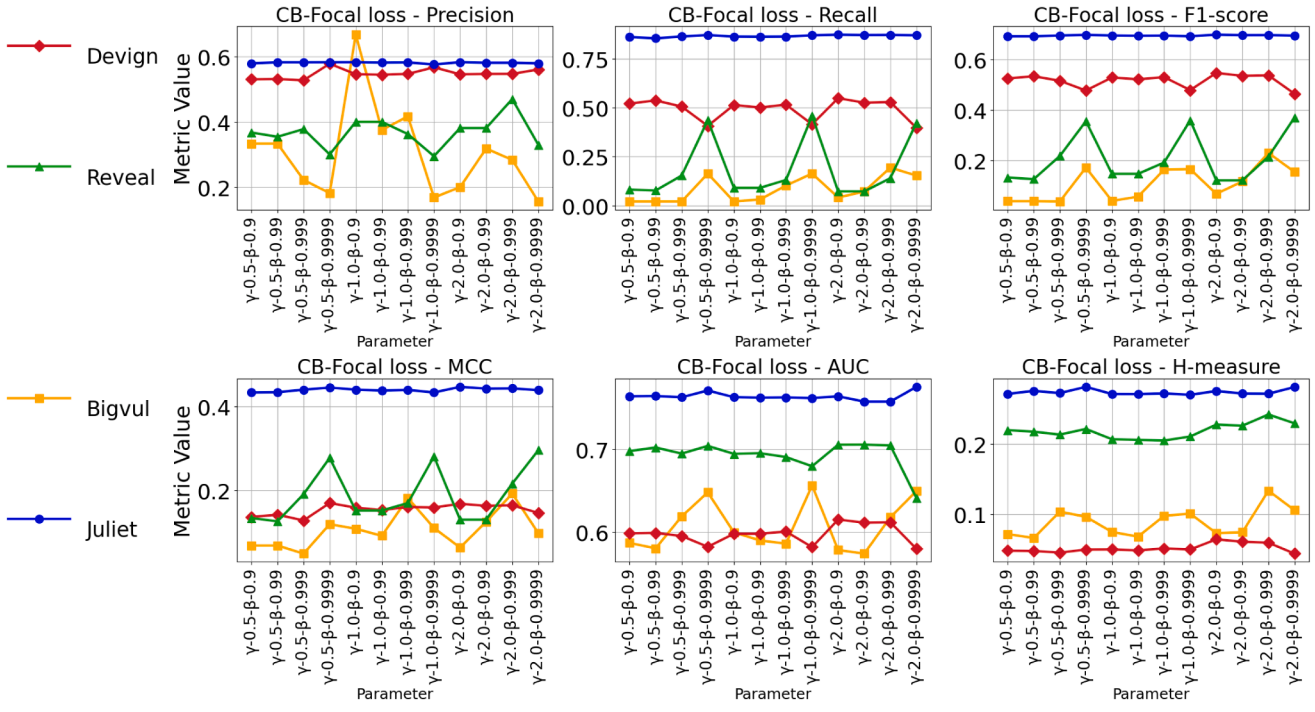
Fig. 12. The parameter exploration of LA loss for all datasets in the LineVul and ReVeal model.

overlap is alleviated to some extent after using imbalance loss functions (i.e., CB-Focal loss and WCE loss).

To further investigate the differences between the CE loss and imbalance loss functions in the feature space, we calculate both inter-class and intra-class distances. This enables a quantitative assessment of the separability between the two classes as well as the degree of clustering within each class. For instance, when LineVul is employed as the vulnerability detection model on the Reveal dataset, the imbalance-aware CB-Focal loss increases the inter-class distance from 44.8577 to 48.8466 and decreases the intra-class distance from 42.3062 to 39.6975, compared to CE loss. Similarly, on the Bigvul dataset, CB-Focal loss raises the inter-class distance from 54.5536 to 60.0694 while reducing the intra-class distance from 36.2806 to 31.6637, demonstrating its effectiveness in enhancing feature separability. Moreover, when ReVeal is adopted as the detection model on the same datasets, the WCE loss function also leads to an increase in inter-class distance. Although it causes a slight rise in intra-class distance (i.e., 0.9208 for the Reveal dataset and 1.9827 for the Bigvul dataset), this increase is relatively minor compared to the



(a) LineVul



(b) ReVeal

Fig. 13. The parameter exploration of CB-Focal loss for all datasets in the LineVul and ReVeal model.

significant enhancement in inter-class distance (i.e., 12.1818 for the Reveal dataset and 12.4969 for the Bigvul dataset), indicating that WCE loss still effectively improves the model's ability to distinguish between vulnerable and non-vulnerable code snippets in the feature space. These result indicates that the imbalance loss functions effectively prioritize the minority class (vulnerable code snippets), allowing the model to better capture its distinguishing features. As a consequence, this leads to better class separation and clustering in the feature space, where samples from the same class are drawn closer together, while those from different classes are pushed farther apart, ultimately improving vulnerability detection performance. This experiment is highly consistent with the findings in RQ1, where imbalance loss functions that achieve the best performance also significantly increase inter-class distance in vulnerability detection, further demonstrating the crucial role of enhanced

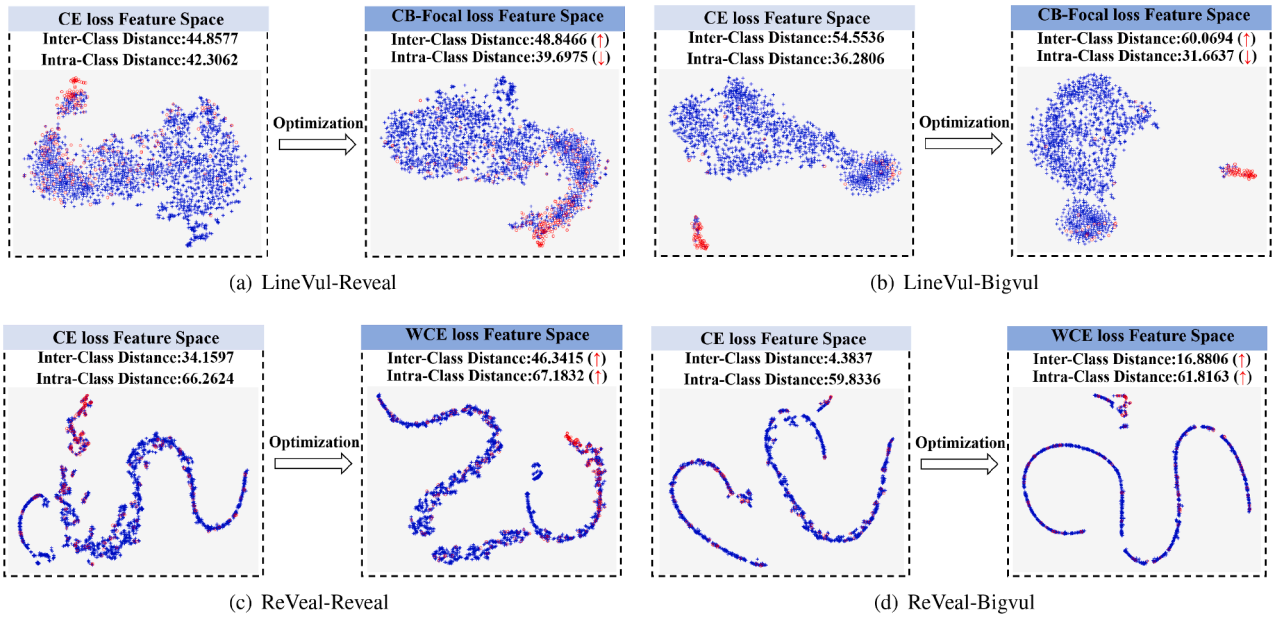


Fig. 14. Feature space visualization of LineVul and ReVeal models on two highly imbalance datasets, comparing CE loss with imbalance loss functions.

linear separability in the feature space for improving model effectiveness. For instance, as shown in Fig. 14(a), on the Reveal dataset, CB-Focal loss increases the inter-class distance and reduces the intra-class distance compared to CE loss. Consistently, CBFocal loss also achieves higher Recall and F1-score in RQ1, further indicating that improving feature separability contributes to better model performance.

7. Implications

Based on our experimental results, we provide several recommendations to offer practical insights and guidance for both researchers and practitioners.

(1) We recommend practitioners and researchers to use the LineVul model or vulnerability detection. Our experimental results demonstrate that the LineVul model outperforms ReVeal across all datasets, with the most significant advantages observed in extremely imbalanced datasets. This indicates that the LineVul model is more effective at detecting vulnerable code snippets while exhibiting better generalizability.

(2) We recommend that practitioners and researchers utilize imbalance loss functions to mitigate the effects of class imbalance, thereby enhancing model performance. In Section 5.1, we compare model performance with nine imbalance loss functions and CE loss across four vulnerability datasets. The results demonstrate that imbalance loss functions such as LDAM loss, LA loss, CB-CE loss, CB-Focal loss, and WCE loss effectively address the impact of class imbalance on DLVD model performance in most cases. For different performance goals, we propose the following strategies to deal with the problem of class imbalance. In tasks that pursue high Precision, it is recommended to combine LDAM loss with two DLVD models to effectively mitigate the impact of class imbalance. For tasks that aim to improve Recall, it is recommended to use the combination of LA loss and LineVul model, and the combination of CB-CE loss and ReVeal model. In the case of pursuing high comprehensive performance (i.e., F1-score, MCC, AUC, and H-measure), we recommend applying CB-Focal loss to these two DLVD models to alleviate the negative impact of class imbalance on performance.

(3) We recommend that practitioners and researchers choose the most appropriate imbalance loss function based on their task objectives. Our experimental results suggest using LDAM loss with LineVul when Precision is a priority, as it enhances Precision across all datasets. Conversely, LA loss is better suited for scenarios where Recall is emphasized. For high overall performance (i.e., high F1-score, MCC, AUC, and H-measure) on extremely imbalanced datasets, CB-Focal loss is recommended. On nearly balanced datasets, LA loss and WCE loss are advisable for achieving strong overall performance. These findings underscore the importance of selecting imbalance loss functions that align with specific performance goals, whether maximizing Precision, Recall, or comprehensive performance.

8. Threats to validity

Construct validity. To ensure the validity and reliability of our results, we utilized rigorous methodologies during our experiments. To mitigate potential bias, we selected two representative DLVD models, LineVul and ReVeal, along with nine imbalance loss functions. While we acknowledge the existence of other DLVD models, this study is not focused on comparing their performance; rather, it aims to explore the effectiveness of imbalance loss functions within the DLVD domain. Additionally, we recognize that some imbalance loss functions remain unexamined, and we plan to investigate the impact of these functions on DLVD models in future research. To minimize technical errors, we utilize established third-party Python libraries such as scikit-learn[9] and PyTorch[10] for implementing our model training code.

External validity. External validity refers to whether the conclusions we draw from our research are applicable to other datasets or actual situations. Our findings are based on cleaned versions of four vulnerability datasets provided by Roland et al.: Devign, Bigvul, Juliet, and the unprocessed Reveal dataset. These datasets have been widely utilized in prior research (Cao et al., 2023; Croft et al., 2023; Wen et al., 2023), and we are optimistic about their relevance. Moving forward, we plan to explore how incorporating additional datasets may further enhance our understanding of the studied loss functions.

Internal validity. A key aspect of the internal validity of the DLVD model is related to the hyperparameter settings during the training phase. Tuning these hyperparameters can be resource-intensive, particularly given the millions of parameters involved in the models. To address this challenge, we adopt the parameter settings recommended in previous studies and reproduce the code provided by the original authors,

[9] https://github.com/scikit-learn/scikit-learn
[10] https://pytorch.org/

strictly adhering to the experimental setup outlined in their paper. This approach enhances the reliability and consistency of our findings. Moreover, we configure the parameters of the imbalance loss functions based on previous studies (Cao et al., 2019; Cui et al., 2019; Li et al., 2019; Lin et al., 2017b; Menon et al., 2021), taking into account the vulnerability percentage of code snippets in each dataset. In future research, we intend to investigate the impact of tuning the hyperparameters of these imbalance loss functions on the performance of the DLVD models.

9. Conclusion and future work

Previous research has largely overlooked the use of imbalance loss functions for training DLVD models to mitigate the adverse effects of class imbalance on model performance. To address this gap, our paper presents the first comprehensive study evaluating the impact of imbalance loss functions on two DLVD models aimed at mitigating class imbalance. We systematically analyze model performance with nine commonly used imbalance loss functions alongside CE loss across four datasets, employing the Scott-Knott ESD test for statistical evaluation of model performance based on six evaluation metrics. Our experimental results yield three key findings: First, the LineVul model consistently outperforms the ReVeal model across all datasets. Second, we demonstrate that the use of certain imbalance loss functions significantly mitigates the effects of class imbalance on DLVD model performance. Third, our analysis indicates that LDAM loss is particularly effective for achieving high Precision, while LA loss is better suited for maximizing Recall. Additionally, CB-Focal loss excels in overall performance on extremely imbalanced datasets, whereas LA loss is advisable for optimizing performance on nearly balanced datasets. In summary, we recommend utilizing the LineVul model in conjunction with either CB-Focal loss or LA loss to achieve optimal performance in DLVD applications.

Since our paper focuses on widely used loss functions, future work will explore additional loss functions to enhance the generalizability of our findings. In addition, we plan to investigate integrating the optimal loss function into vulnerability detection models for practical deployment in industrial-grade code auditing systems, aiming to automate vulnerability detection in software repositories and validate the real-world effectiveness of our approach.

CRediT authorship contribution statement

Xiaoxue Ma: Data curation, Methodology, Formal analysis, Writing - original draft; **Yanzhong He:** Resources, Software, Visualization, Writing - original draft; **Jacky Keung:** Formal analysis, Writing - review & editing, Validation; **Cheng Tan:** Project administration, Software, Validation; **Chuanxiang Ma:** Resources, Software, Visualization, Writing - review & editing; **Wenhua Hu:** Formal analysis, Methodology, Validation; **Fuyang Li:** Supervision, Methodology, Validation, Writing - review & editing.

Data availability

No data was used for the research described in the article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (No. 62202350), the Major Science and Technology Project of Hubei Province (No. 2024BAA008), the General Research Fund of the Research Grants Council of Hong Kong and the research funds of the City University of Hong Kong (No. 6000796, No. 9229109, No. 9229098, No. 9220103, and No. 9229029), the Start-up Grant from Wuhan University of Technology (No. 104–Wuhan University of Technology 40120693), and the National Training Program of Innovation and Entrepreneurship for Undergraduates (No. 202410497045).

Supplementary material

Supplementary material associated with this article can be found in the online version at 10.1016/j.eswa.2025.128504

References

Aftabi, N., Moradi, N., Mahroo, F., & Kianfar, F. (2025). Sd-abm-ism: An integrated system dynamics and agent-based modeling framework for information security management in complex information systems with multi-actor threat dynamics. *Expert Systems with Applications*, *263*, 125681.

Alabdulwahab, S., Cheong, M., Seo, A., Kim, Y.-T., & Son, Y. (2024). Enhancing deep learning-based side-channel analysis using feature engineering in a fully simulated iot system. *Expert Systems with Applications*, *266*, (p. 126079).

Boland, T., & Black, P. E. (2012). Juliet 1. 1 c/c++ and java test suite. *Computer*, *45*(10), 88–90.

Cao, K., Wei, C., Gaidon, A., Aréchiga, N., & Ma, T. (2019). Learning imbalanced datasets with label-distribution-aware margin loss. In *Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019* (pp. 1565–1576).

Cao, S., Sun, X., Bo, L., Wu, R., Li, B., Wu, X., Tao, C., Zhang, T., & Liu, W. (2023). Learning to detect memory-related vulnerabilities. *ACM Transactions on Software Engineering and Methodology*, *33*(2), 1–35.

Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2022). Deep learning based vulnerability detection: Are we there yet. *IEEE Transactions on Software Engineering*, *48*(9), 3280–3296.

Cheng, X., Wang, H., Hua, J., Xu, G., & Sui, Y. (2021). Deepwukong: Statically detecting software vulnerabilities using deep graph neural network. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *30*(3), 1–33.

Croft, R., Babar, M. A., & Kholoosi, M. M. (2023). Data quality for software vulnerability datasets. In *45th IEEE international conference on software engineering* (pp. 121–133). IEEE.

Cui, Y., Jia, M., Lin, T., Song, Y., & Belongie, S. J. (2019). Class-balanced loss based on effective number of samples. In *IEEE conference on computer vision and pattern recognition* (pp. 9268–9277).

Dipongkor, A. K., & Moran, K. (2023). A comparative study of transformer-based neural text representation techniques on bug triaging. In *38th IEEE international conference on automated software engineering* (pp. 1012–1023).

Dong, Y., Tang, Y., Cheng, X., Yang, Y., & Wang, S. (2023). SedSVD: Statement-level software vulnerability detection based on relational graph convolutional network with subgraph embedding. *Information and Software Technology*, *158*, 107168.

Fan, J., Li, Y., Wang, S., & Nguyen, T. N. (2020). A c/c++ code vulnerability dataset with code changes and CVE summaries. In *17th international conference on mining software repositories* (pp. 508–512).

Fu, M., & Tantithamthavorn, C. (2022). Linevul: A transformer-based line-level vulnerability prediction. In *19th mining software repositories* (pp. 608–620).

Hanif, H., & Maffeis, S. (2022). Vulberta: Simplified source code pre-training for vulnerability detection. In *2022 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

He, Y., Lin, G., Ma, X., Keung, J. W., Tan, C., Hu, W., & Li, F. (2024). Enhancing deep learning vulnerability detection through imbalance loss functions: An empirical study. In *Proceedings of the 15th Asia-Pacific symposium on internetware* (pp. 85–94).

Islam, N. T., Torre, P. G. D. L., Manuel, D., Bou-Harb, E., & Najafirad, P. (2023). An unbiased transformer source code learning with semantic vulnerability graph. In *2023 IEEE 8th European symposium on security and privacy (euros&p)* (pp. 144–159). IEEE.

Jiang, Y., Zhang, Y., Su, X., Treude, C., & Wang, T. (2024). StagedvulBERT: Multi-granular vulnerability detection with a novel pre-trained code model. *IEEE Transactions on Software Engineering*, *50*(12), 3454–3471.

Li, B., Liu, Y., & Wang, X. (2019). Gradient harmonized single-stage detector. In *Proceedings of 33rd AAAI conference on artificial intelligence* (pp. 8577–8584). (*vol. 33*).

Li, F., Zou, K., Keung, J. W., Yu, X., Feng, S., & Xiao, Y. (2023). On the relative value of imbalanced learning for code smell detection. *Software: Practice and Experience*, *53*(10), 1902–1927.

Li, X., Sun, X., Meng, Y., Liang, J., Wu, F., & Li, J. (2020). Dice loss for data-imbalanced NLP tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics, ACL 2020, online, july 5–10, 2020* (pp. 465–476). Association for Computational Linguistics.

Li, Y., Wang, S., & Nguyen, T. N. (2021a). 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* (pp. 292–303).

Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2021b). Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing*, *19*(4), 2244–2258.

Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., & Zhong, Y. (2018). Vuldeepecker: A deep learning-based system for vulnerability detection. In *25th annual network and distributed system security symposium*.

Lin, G., Jia, H., & Wu, D. (2022). Distilled and contextualized neural models benchmarked for vulnerable function detection. *Mathematics*, *10*(23), 4482.

Lin, G., Xiao, W., Zhang, L. Y., Gao, S., Tai, Y., & Zhang, J. (2021). Deep neural-based vulnerability discovery demystified: Data, model and performance. *Neural Computing and Applications*, *33*(20), 13287–13300.

Lin, G., Zhang, J., Luo, W., Pan, L., De Vel, O., Montague, P., & Xiang, Y. (2019). Software vulnerability discovery via learning multi-domain knowledge bases. *IEEE Transactions on Dependable and Secure Computing*, *18*(5), 2469–2485.

Lin, G., Zhang, J., Luo, W., Pan, L., & Xiang, Y. (2017a). Poster: Vulnerability discovery with function representation learning from unlabeled projects. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security* (pp. 2539–2541).

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017b). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980–2988).

Liu, H., Jiang, S., Qi, X., Qu, Y., Li, H., Li, T., Guo, C., & Guo, S. (2024a). Detect software vulnerabilities with weight biases via graph neural networks. *Expert Systems with Applications*, *238*, 121764.

Liu, S., Ma, W., Wang, J., Xie, X., Feng, R., & Liu, Y. (2024b). Enhancing code vulnerability detection via vulnerability-preserving data augmentation. In *Proceedings of the 25th ACM SIGPLAN/SIGBED international conference on languages, compilers, and tools for embedded systems* (pp. 166–177).

Ma, X., Zou, H., He, P., Keung, J., Li, Y., Yu, X., & Sarro, F. (2024). On the influence of data resampling for deep learning-based log anomaly detection: Insights and recommendations. *IEEE Transactions on Software Engineering*, *51*(1), 243–261.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11).

Menon, A. K., Jayasumana, S., Rawat, A. S., Jain, H., Veit, A., & Kumar, S. (2021). Long-tail learning via logit adjustment. In *9th international conference on learning representations*.

Milletari, F., Navab, N., & Ahmadi, S.-A. (2016). V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3d vision (3DV)* (pp. 565–571). IEEE.

Nguyen, T., Le, T., Nguyen, K., Vel, O. d., Montague, P., Grundy, J., & Phung, D. (2020). Deep cost-sensitive kernel machine for binary software vulnerability detection. In *Advances in knowledge discovery and data mining: 24th Pacific-Asia conference, PAKDD 2020, singapore, may 11–14, 2020, proceedings, part II 24* (pp. 164–177). Springer.

Nong, Y., Ou, Y., Pradel, M., Chen, F., & Cai, H. (2023). VULGEN: Realistic vulnerability generation via pattern mining and deep learning. In *45th international conference on software engineering* (pp. 2527–2539).

Qiu, F., Liu, Z., Hu, X., Xia, X., Chen, G., & Wang, X. (2024). Vulnerability detection via multiple-graph-based code representation. *IEEE Transactions on Software Engineering*, *50*(8), 2178–2199.

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, *1*, 127–190.

Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., & McConley, M. (2018). Automated vulnerability detection in source code using deep representation learning. In *17th IEEE international conference on machine learning and applications* (pp. 757–762).

Song, Z., Wang, J., Yang, K., & Wang, J. (2023). Hgivul: Detecting inter-procedural vulnerabilities based on hypergraph convolution. *Information and Software Technology*, *160*, 107219.

Steenhoek, B., Gao, H., & Le, W. (2024). Dataflow analysis-inspired deep learning for efficient vulnerability detection. In *Proceedings of the 46th IEEE/ACM international conference on software engineering* (pp. 1–13).

Steenhoek, B., Rahman, M. M., Jiles, R., & Le, W. (2023). An empirical study of deep learning models for vulnerability detection. In *2023 IEEE/ACM 45th international conference on software engineering (ICSE)* (pp. 2237–2248). IEEE.

Tang, M., Tang, W., Gui, Q., Hu, J., & Zhao, M. (2024). A vulnerability detection algorithm based on residual graph attention networks for source code imbalance (RGAN). *Expert Systems with Applications*, *238*, 122216.

Tang, W., Tang, M., Ban, M., Zhao, Z., & Feng, M. (2023). Csgvd: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. *Journal of Systems and Software*, *199*, 111623.

Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2018). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering*, *46*(11), 1200–1219.

Tenney, I., Wexler, J., Bastings, J., Bolukbasi, T., Coenen, A., Gehrmann, S., Jiang, E., Pushkarna, M., Radebaugh, C., Reif, E. et al. (2020). The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. In *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations* (pp. 107–118).

Tian, Z., Tian, B., Lv, J., Chen, Y., & Chen, L. (2024). Enhancing vulnerability detection via AST decomposition and neural sub-tree encoding. *Expert Systems with Applications*, *238*, 121865.

Wang, Q., Li, Z., Liang, H., Pan, X., Li, H., Li, T., Li, X., Li, C., & Guo, S. (2024a). Graph confident learning for software vulnerability detection. *Engineering Applications of Artificial Intelligence*, *133*, 108296.

Wang, R., Xu, S., Tian, Y., Ji, X., Sun, X., & Jiang, S. (2024b). Scl-cvd: Supervised contrastive learning for code vulnerability detection via graphcodebert. *Computers & Security*, *145*, 103994.

Wang, S., Huang, C., Yu, D., & Chen, X. (2023a). Vulgrab: Graph-embedding-based code vulnerability detection with bi-directional gated graph neural network. *Software: Practice and Experience*, *53*(8), 1631–1658.

Wang, W., Nguyen, T. N., Wang, S., Li, Y., Zhang, J., & Yadavally, A. (2023b). DeepVD: Toward class-separation features for neural network vulnerability detection. In *45th international conference on software engineering* (pp. 2249–2261).

Wei, H., Lin, G., Li, L., & Jia, H. (2021). A context-aware neural embedding for function-level vulnerability detection. *Algorithms*, *14*(11), 335.

Wen, X.-C., Chen, Y., Gao, C., Zhang, H., Zhang, J. M., & Liao, Q. (2023). Vulnerability detection with graph simplification and enhanced graph representation learning. In *2023 IEEE/ACM 45th international conference on software engineering (ICSE)* (pp. 2275–2286). IEEE.

Wen, X.-C., Gao, C., Gao, S., Xiao, Y., & Lyu, M. R. (2024). Scale: Constructing structured natural language comment trees for software vulnerability detection. In *Proceedings of the 33rd ACM SIGSOFT international symposium on software testing and analysis* (pp. 235–247).

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (pp. 1–10).

Xu, L., An, B., Li, X., Zhao, D., Peng, H., Song, W., Tong, F., & Han, X. (2024). Tfhsvul: A fine-grained hybrid semantic vulnerability detection method based on self-attention mechanism in iot. *IEEE Internet of Things Journal*, *12*(1), 30–44.

Yang, X., Wang, S., Li, Y., & Wang, S. (2023). Does data sampling improve deep learning-based vulnerability detection? yeas! and nays! In *45th international conference on software engineering* (pp. 2287–2298).

Yu, X., Keung, J., Xiao, Y., Feng, S., Li, F., & Dai, H. (2022). Predicting the precise number of software defects: Are we there yet? *Information and Software Technology*, *146*, 106847.

Yu, X., Lin, G., Hu, X., Keung, J. W., & Xia, X. (2025). Less is more: Unlocking semi-supervised deep learning for vulnerability detection. *ACM Transactions on Software Engineering and Methodology*, *34*(3), 1–37.

Zhang, C., Liu, B., Xin, Y., & Yao, L. (2023a). Cpvd: Cross project vulnerability detection based on graph attention network and domain adaptation. *IEEE Transactions on Software Engineering*, *49*(8), 4152–4168.

Zhang, F., Zhang, Z., Keung, J. W., Tang, X., Yang, Z., Yu, X., & Hu, W. (2024). Data preparation for deep learning based code smell detection: A systematic literature review. *Journal of Systems and Software*, *216*, (p. 112131).

Zhang, J., Liu, Z., Hu, X., Xia, X., & Li, S. (2023b). Vulnerability detection by learning from syntax-based execution paths of code. *IEEE Transactions on Software Engineering*, *49*(8), 4196–4212.

Zhou, Y., Liu, S., Siow, J. K., Du, X., & Liu, Y. (2019). Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019* (pp. 10197–10207).

Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., & Xu, B. (2018). How far we have progressed in the journey? an examination of cross-project defect prediction. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *27*(1), 1–51.