Does data sampling improve deep learning-based vulnerability detection? Yeas! and Nays!

Xu Yang, Shaowei Wang
University of Manitoba, Canada
yangx4@myumanitoba.ca, shaowei.wang@umanitoba.ca

Yi Li, Shaohua Wang
New Jersey Institute of Technology, USA
yl622@njit.edu, davidshwang@ieee.org

2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00192

***Abstract*—Recent progress in Deep Learning (DL) has sparked interest in using DL to detect software vulnerabilities automatically and it has been demonstrated promising results at detecting vulnerabilities. However, one prominent and practical issue for vulnerability detection is data imbalance. Prior study observed that the performance of state-of-the-art (SOTA) DL-based vulnerability detection (DLVD) approaches drops precipitously in real world imbalanced data and a 73% drop of F1-score on average across studied approaches. Such a significant performance drop can disable the practical usage of any DLVD approaches. Data sampling is effective in alleviating data imbalance for machine learning models and has been demonstrated in various software engineering tasks. Therefore, in this study, we conducted a systematical and extensive study to assess the impact of data sampling for data imbalance problem in DLVD from two aspects: i) the effectiveness of DLVD, and ii) the ability of DLVD to reason correctly (making a decision based on real vulnerable statements). We found that in general, oversampling outperforms undersampling, and sampling on raw data outperforms sampling on latent space, typically random oversampling on raw data performs the best among all studied ones (including advanced one SMOTE and OSS). Surprisingly, OSS does not help alleviate the data imbalance issue in DLVD. If the recall is pursued, random undersampling is the best choice. Random oversampling on raw data also improves the ability of DLVD approaches for learning real vulnerable patterns. However, for a significant portion of cases (at least 33% in our datasets), DVLD approach cannot reason their prediction based on real vulnerable statements. We provide actionable suggestions and a roadmap to practitioners and researchers.**

***Index Terms*—Vulnerability detection, deep learning, data sampling, interpretable AI**

I. Introduction

Machine learning-based VD approaches have attracted more attention from research community since they can learn vulnerability patterns from prior vulnerable code automatically [1]–[5]. Especially, recent progress in Deep Learning (DL) has sparked interest in using DL to detect software vulnerabilities automatically. In fact, recent studies have demonstrated very promising results achieving high accuracy in detecting vulnerabilities [3]–[5].

The prominent and practical issue for vulnerability detection is data imbalance. The ratio of vulnerable and non-vulnerable cases in real-world projects is extremely unbalanced. Vulnerable cases are far fewer than non-vulnerable ones. Prior study [3] observed that the performance of state-of-the-art (SOTA) DL-based vulnerability detection (DLVD) approaches drops precipitously in real world unbalanced data. On average, a 73% drop of F1 across all the models is observed. Such a significant performance drop can disable the practical usage of any DLVD approaches. Various approaches were developed to deal with the problem, such as data sampling, cost sensitive learning, and ensemble methods. Data sampling is a widely-used technique and proved to help solve the data imbalance issues in Software Engineering tasks, such as defect prediction [6], [7], software quality prediction [8], and software change prediction [9]. Despite the prevalent existence of imbalance issue in DLVD, no research that systemically and extensively studies data sampling for data imbalance problem in DLVD has been done. Thus, it is in dire need to understand data sampling for data imbalance problem in DLVD.

In this paper, we aim to assess the impact of data sampling on the effectiveness of existing SOTA DLVD approaches and their ability of learning vulnerable patterns. For this goal, we conducted an extensive study on four data sampling approaches (random under/oversampling, SMOTE [10], and OSS [11]), and their impact on four SOTA DLVD approaches: Devign [12], Reveal [3], IVDetect [5], and LineVul [13], using three benchmark datasets. Note that some advanced data sampling approaches could only be applied to the data points that have been projected into latent space (feature space) since they require computation in the latent space, such as the popular one SMOTE [10]. While simple ones, such as random under/oversampling, can be applied both on raw data (without any projection) and on latent space. On one hand, latent space sampling is cheaper than sampling on raw data, since it saves the resource and time for preprocessing data (e.g., data cleanup and feature extraction) and training models. On the other hand, projection into latent space causes information loss. It is unknown which strategy is better. Therefore, we also investigate those two sampling strategies - sampling on raw data (i.e., raw code without any pre-processing and feature extraction) and sampling on latent space (i.e., representation vectors of code). We formulate our research as the following research questions:

- **RQ1: Does data sampling improve the effectiveness of existing DLVD approaches?**
  We applied various data sampling approaches to the state-of-the-art DLVD approaches and compared them with the ones without sampling. **Results:** Generally, oversampling

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00192

outperforms undersampling, and sampling on raw data outperforms sampling on latent space. Simple approach random oversampling on raw data beats all other studied ones including advanced one SMOTE and OSS. Surprisingly, OSS does not help alleviate the data imbalance problem in DLVD if the DLVD approaches perform poorly on imbalanced data originally. Random undersampling performs the best in improving recall on imbalanced datasets.

- **RQ2: Does data sampling improve the ability of DLVD for learning the vulnerable patterns?**
  We aim to understand if a trained DL model with sampling approaches can make more prediction decision reasoning over real vulnerable statements in functions than the same DL model without sampling approaches. We used interpretable AI techniques (LIME [14] and GNNExplainer [15]) to interpret the predictions made by DLVD approaches and examine whether their decisions are made based on the real vulnerable patterns (i.e., vulnerable statements). **Results:** Random oversampling improves the ability of DLVD approaches for learning real vulnerable patterns. However, in a significant portion of cases (at least 33% in our studied datasets), DVLD approaches cannot reason their prediction based on real vulnerable statements. There is still room for improving the ability of learning real vulnerable patterns for DLVD approaches.

In summary, the contributions of our paper include:

- To our best knowledge, we conducted the first systematic and extensive study to assess the impact of data sampling for data imbalance problem in DLVD. We study four data sampling approaches and two data sampling strategies and their impact on four SOTA DLVD approaches using three benchmark datasets. We ran 1,680 experiments, costing more than 10,200 GPU hours.
- We provide actionable suggestions and a roadmap to practitioners and researchers for DLVD. **Yeas1:** oversampling is recommended over undersampling. **Yeas2:** sampling on raw data is recommended over sampling on latent space. **Yeas3:** random oversampling on raw data is recommended to handle data imbalance problem in DLVD compared. **Yeas4:** future research is suggested to develop new data augmentation to improve the ability of DLVD approaches for learning real vulnerable patterns, as there is still room to improve. **Nays1:** OSS is not recommended to handle data imbalance problem in DLVD.
- A replication package https://github.com/WIP2022/DataSampling4DLVD for future research and improvement.

II. Background

In this section, we introduce the background related to deep learning-based vulnerability detection and data sampling.

A. Overview of deep learning-based vulnerability detection

In general, DLVD approaches consist of three phases: feature extraction, model training, and model deployment. Figure 1 presents the general framework of DLVD approaches in a green rectangle. First, in the feature extraction phase, various features from code units that could well capture the semantic and syntactic properties of code to discriminate vulnerable code from non-vulnerable code are extracted. Second, the extracted features are transformed into a real-valued vector, which is a compact representation of code units (i.e., *representation learning*). What features are extracted and how a representation is learned depend on the techniques being selected. Once the representation learning is done, a binary classification model is selected to perform vulnerability detection. We discuss more details on specific representation learning in Section II-B. In the deployment phase, the same features extracted in the training phase are extracted and the same representation learning techniques are applied to the code unites of target software systems to generate the representation vector.

B. Existing DL-based vulnerability detection approaches

Generally, DLVD approaches differ in two aspects: the types of features extracted from a code unit (i.e., feature extraction) and the way of transforming the extracted features into a representation vector (i.e., representation learning). Based on this, DLVD approaches could be categorized into two families: token-based or graph-based approaches.

Token-based: In the token-based approach, code is considered as a sequence of tokens and is represented as a vector via text embedding techniques (e.g., Word2Vec [16],GloVe [17]). For instance, LineVul [13] leverages CodeBERT [18] to embed whole sequence of tokens in a function for vulnerability detection. CodeBERT is a transformer-based model pre-trained on a large corpus of source code in various languages and can use the attention mechanism to learn the relationship between tokens. Instead of considering the whole code, approaches such as VulDeePecker [2] and SySeVR [19], extract slices from points of interest in code (e.g., API calls, array indexing, pointer usage, etc.) and use them for vulnerability detection since they assume that different lines of code are not equivalently important for vulnerability detection.

Graph-based: Besides considering the semantic information represented in the sequential tokens, graph-based approaches also consider code as graphs and incorporate the information in different syntactic and semantic dependencies using graph neural network (GNN) [20] when generating the representation vectors. Different types of syntactic/semantic graphs, (e.g., abstract syntax tree (AST), program dependency graph (PDG), code property graph (CPG)) can be used. For example, Devign [12] and Reveal [3] leverage code property graph (CPG) [21] to build their graph-based vulnerability detection model. They both use the gated graph neural network (GGNN) to represent the code. IVDetect [5] and LineVD [4] consider the vulnerable statements and capture their surrounding contexts via program dependency graph. IVDetect uses Feature-Attention GCN model(FA-GCN) for graph embedding and LineVD use graph attention network (GAT) model for graph embedding. Differently, IVDetect performs function-level detection while LineVD performs line-level.

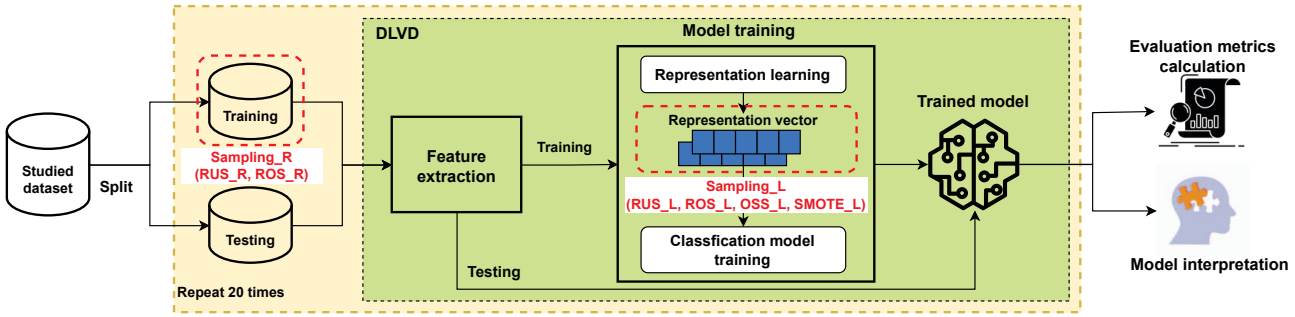

Fig. 1: Overview of our approach. The deep learning-based vulnerability detection framework is presented in the green rectangle. We mark steps where the proper data sampling approaches could be performed in red round-rectangles.

TABLE I: Prior approaches using deep learning for vulnerability detection.

Approach	Extracted feature	Representation learning	Classifier	Detection level
LineVul [13]	tokens	CodeBERT	MLP	Function/line
VulDeePecker [2]	tokens + code slices	word2vec	BLSTM	Function
SySeVR [22]	tokens + code slices	word2vec	BGRU	Function
Devign [12]	tokens + CPG	word2vec + GGNN	MLP	Function
Reveal [3]	tokens + CPG	word2vec + GGNN	MLP	Function
IVDetect [5]	tokens + AST+PDG	GloVe + FA-GCN	MLP	Function
LineVD [4]	tokens + PDG	CodeBERT + GAT	MLP	Line

In both graph-based and text-based models, once the representation for code units is learned, the vector will be passed into a classfier for final learning. Various classifiers have been investigated in prior studies. For instance, IVDetect [5] and Reveal [3] use Multilayer perceptron (MLP) as the classification model. Table I summarizes the framework of the above-mentioned DLVD approaches.

C. Data sampling

In general, data sampling approaches could be categorized into two families: oversampling and undersampling.

Undersampling vs. oversampling Undersampling refers to deleting data points from the majority class of the dataset so that the remaining majority class is approximately equal in number to the minority class [23]. In the context of vulnerability detection, undersampling deletes a certain number of non-vulnerable cases from the training data so that the ratio of vulnerable/non-vulnerable cases approximately equals 1. For instance, random undersampling (RUS) randomly removes cases from the majority class, with or without replacement. Tomek Links locates all cross-class nearest neighbor pairs and remove the corresponding cases in the majority class that are closest to the minority class [24]. One-sided selection (OSS) combines Tomek Links and the Condensed Nearest Neighbor (CNN) Rule [11]. However, undersampling is very likely to discard useful or important samples [23] and cause information loss.

Oversampling increases the number of instances in the minority class to balance the dataset [23]. For instance, random oversampling (ROS) randomly selects cases from the minority class, with replacement, and adding them to the training dataset. ROS has been proven to be robust [25]. One limitation of ROS is that it does not provide any additional information to the model. Another popular oversampling technique called Synthetic Minority Oversampling Technique (SMOTE) [23] was developed to address this issue by augmenting data for the minority class. It first selects a minority class instance a at random and finds its nearest majority class neighbor b. The synthetic instance is then created by choosing one point on the line connecting a and b in the feature space.

Raw data sampling vs. latent space sampling Some advanced data sampling approaches could only be applied to the data points that have been projected into latent space (or feature space) since they require computation (e.g., calculating distance between points) in the latent space, such as SMOTE. While some simple ones, such as RUS and ROS, can be done on the raw data (raw code units in our case) without any projection into latent space. For simplicity, we refer to the sampling performed on the raw data without any further process as **sampling_R** and the sampling performed on data in the latent space as **sampling_L**. Based on the definition, random oversampling and random undersampling could be performed both on raw data and latent space, while SMOTE and OSS could only be on latent space. In the context of DLVD, we can perform sampling_R on the raw training data and perform sampling_L on representation vectors after representation learning as indicated in Figure 1. On one hand, sampling_L is cheaper than sampling_R, since sampling_L saves the resource and time to preprocess (e.g., data cleanup, feature extraction) and train the backbone model on the extra instances that are added in the minority class. On other hand, projection into latent space cause information loss. We compare them in our RQs. For simplicity, we refer to the same sampling approach applied to raw data and latent space as two approaches in the rest of the paper. For instance, we refer to random oversampling on latent space as ROS_L and on raw

data as ROS_R.

III. Experimental design

In this section, we present our research questions (RQs), our studied dataset, DLVD approaches, data sampling approaches, and our analysis approach for RQs.

A. Research questions

We aim to answer the following research questions:

- **RQ1: Does data sampling improve the effectiveness of existing DLVD approaches?**
- **RQ2: Does data sampling improve the ability of DLVD for learning the vulnerable patterns?**

Data sampling has been shown its effectiveness in alleviating data imbalance issues in various software engineering tasks, such as defect prediction [6], [7] and quality prediction [8]. Little research was on the impact of data sampling on the performance of SOTA DLVD approaches. **In RQ1**, we investigated whether data sampling could improve the effectiveness of existing SOTA DLVD approaches. Through RQ1, we could provide practitioners insights on the selection of proper data sampling approaches based on their context. Prior research has shown that even if a DL model is able to correctly predict an instance, the prediction is not always based on the real pattern [3], [14]. For instance, in the vulnerability detection context, suppose a DLVD approach correctly predicts a function as vulnerable. However, the statements that the model reasons to make the decision are not the real vulnerable statements in the function. In other words, the model does not really learn the vulnerable patterns for discriminating from vulnerable code to non-vulnerable code. Therefore, **in RQ2**, we aim to understand whether data sampling could help a DLVD approach improve its ability to learn real vulnerable patterns.

B. Datasets

To answer our research questions, we conducted our study on three popular vulnerability datasets including BigVul [26], Reveal [3], and Devign [12]. BigVul [26] covers the CWEs from 2002 to 2019 that are extracted from over 300 different open source C/C++ projects and contains the trustworthy source code vulnerabilities spanning 91 different vulnerability types. It contains +10K vulnerable methods and +160K non-vulnerable methods. The Reveal dataset [3] contains +12K methods with 9.16% of the vulnerable ones. The Devign dataset [12] has +22K methods collected from projects FFM-Peg and Qemu, in which 45.0% of the methods are vulnerable. The Devign dataset is balanced compared with another two datasets. Those datasets are widely used to evaluate various DLVD approaches in prior studies [3]–[5], [12], [13]. Note that only the BigVul dataset has vulnerability-fixing information.

TABLE II: Overview of the studied datasets.

Dataset	#Vulnerablities	#Non-Vulnerabilities	Ratio
Reveal [3]	1,664	10,547	1:9.9
Devign [12]	10,067	12,294	1:1.2
BigVul [26]	10,547	168,752	1:16.3

C. Studied DLVD & Data sampling approaches

DLVD. We chose DLVD approaches based on two criteria. First, DLVD should be representative of the two families of DLVD approaches, token-based and graph-based approaches as discussed in Section II-B. Second, the approaches should be recently developed and leverage deep learning techniques. The goal behind the criterion is to foster the generalizability and applicability of our results to SOTA approaches. We selected Devign [12], Reveal [3], and IVDetect [5] as the representative for graph-based approaches. For the token-based approach, we selected LineVul [13] as the representative since it is SOTA approach leveraging CodeBERT [18], and has been proven to be more effective than other approaches (e.g., Devign, Reveal, and IVDetect). Note that LineVul can be used to perform both function-level and line-level vulnerability detection. In this study, we used LineVul to perform function-level detection for consistency.

Data sampling. We selected data sampling approaches based on two criteria. First, the selected approach should cover two families of sampling approaches in Section II-C, i.e., oversampling and undersampling. Second, the selected approaches can be applied on both raw data and latent space. Therefore, we selected random undersampling (RUS) and random oversampling (ROS) as our studied approaches. In addition, we also wish to include advanced approaches besides ROS and RUS. We selected one-side selection (OSS) as the representative for undersampling, and Synthetic Minority Oversampling Technique (SMOTE) as the representative for oversampling. Note that OSS and SMOTE could only be applied on latent space as discussed in Section II-C.

D. Evaluation metrics

We consider two families of evaluation metrics to assess the effectiveness of DLVD approaches. First, considering the vulnerability detection as a binary classification task, we consider four popular evaluation metrics [27]- recall (short for **R**), precision (short for **P**), F1, and Area Under The Curve (AUC) following prior studies [2], [3], [12], [28], [29]. Second, considering the cost of reviewing potential vulnerable code from a returned list, we used popular evaluation metric for ranking algorithm [5], [30] - Precision@k (short for **P@k**), which is calculated as $\frac{true\ positive}{k}$ and k is the size of returned list. We selected k to be 10, 20, 50, and 100.

E. Approach of RQs

We first introduce certain settings in our study. We consider the assessment of data sampling approaches of one DLVD approach on one dataset as one *experimental instance*. For simplicity, we use the format $\{DLVD\}+\{Dataset\}$ to denote one specific instance in the rest of the sections. Therefore, We have 12 experimental instances, i.e., 4 DLVD * 3 datasets.

For each experimental instance, we have the following three settings of data sampling to apply and compare their impact on the DLVD approach:

NoSampling: We train the DLVD approach on the original training data without applying any data sampling approach and evaluate the DLVD approach on the testing data.

Sampling_R: We apply data sampling approaches (i.e., RUS and ROS) on the raw training data (shown in Figure 1 in red round-rectangle) and train the DLVD approach on the sampled training data. We then evaluate the DLVD approach on testing data. We refer to RUS and ROS on raw data as RUS_R and ROS_R, respectively.

Sampling_L: This setting is similar to **Sampling_R**. The only difference is that instead of applying data sampling approaches on raw training data, we apply data sampling (i.e., RUS, ROS, OSS, and SMOTE) on representation vectors of training data (latent space). We refer to RUS, ROS, OSS, and SMOTE on latent space as RUS_L, ROS_L, OSS_L, and SMOTE_L, respectively.

Data splitting for generating training and testing data involves a random process, to alleviate the bias caused by the randomness, we repeated the process 20 times following prior studies [31]–[33] and calculated the mean value across the 20 splits for each evaluation metric as our results. We split the training and testing to 80%/20% following prior studies [3], [34], [35].

1) Approach of RQ1: In RQ1, our goal is to examine if data sampling could improve the effectiveness of DLVD and if so, which data sampling approaches perform the best. For this purpose, we rank data sampling approaches in each experimental instance based on our studied evaluation metrics (see Section III-D for details). For example, suppose we apply the studied sampling approaches on the Reveal dataset for training the IVDetect model (i.e., $IVDetect+Reveal$). In this experimental instance, ROS_R performs the best (ranks 1st) in terms of F1, we note that ROS_R ranks 1st in the instance $IVDetect + Reveal$ in terms of F1. To get an overall rank for a data sampling approach across the studied experimental instances, we compute an average rank. For instance, suppose ROS_R ranks 1st in 6 instances and ranks 2nd in another 6 instances, then its average rank is 1.5. Therefore, we can compare data sampling approaches based on their rank in terms of different evaluation metrics. A smaller rank indicates more effective. We also investigated which sampling approach performs the best by comparing the number of experimental instances won by each sampling approach.

2) Approach of RQ2: To understand if the trained model makes correct prediction decision reasoning over real vulnerable statements in a function, we selected to use interpretable AI techniques. The three graph-based models we studied all use GNNs to capture graph-related properties. Hence, we selected the Reveal model as the representative for graph-based approaches and selected LineVul as the representative for text-based approaches. We selected the BigVul dataset since only this dataset provides us the line information for vulnerable codes, i..e, specific statements that make the code vulnerable. Therefore, we conducted the analysis on two experimental instances $LineVul + BigVul$ and $Reveal + BigVul$.

We used Local Interpretable Model-Agnostic Explanations (LIME) [14], which is a widely-used model-agnostic explainable algorithm for explaining deep learning models, to interpret LineVul. LIME returns tokens in a sequence that make important contribution to the prediction. For our case, we compared the results of LIME with the real vulnerable statements in the functions. More specifically, we designed the following experiment for LineVul: we selected the true positive (TP) cases (the vulnerable functions that are predicted correctly) predicted by LineVul in the testing set of BigVul, and analyzed those cases using LIME. In the result of LIME, for each true positive case, we selected the top k most important tokens that drive the model to make the decision and see whether those tokens are inside the vulnerable statements. If at least one of the top k is inside the vulnerable statements, we consider it as a **hit**, and we compare the ratio of **hit** over all TP cases between NoSampling and all sampling approaches. We considered 1, 3, 5, and 10 for k.

We used GNNExplainer [15] to interpret Reveal following prior study [5]. Similarly, we interpreted the true positive (TP) cases predicted by the Reveal model. GNNExplainer provides the importance of each edge in the graph. Reveal uses the code property graph (CPG), and each line of code is a node in the graph, we applied GNNExplainer on the CPG. GNNExplainer does not directly tell us which nodes are important, therefore we consider the nodes connected by the important edges returned by GNNExplainer as the important nodes in the graph. Similar to the experiment of Lime, we selected the top k most important edges and their connected nodes that drive the model to predict a function to be vulnerable and see whether those nodes (lines of codes) are inside the vulnerable statements of the function.

TABLE III: Parameter setting for studied DLVDs.

Parameter	IVDetect	Devign	Reveal	LineVul
Epochs	50	50	50	4
Learning_rate	1e-4	1e-4	1e-4	5e-5
GNN_Layer	4	6	6	NaN
graph embedding size	100	200	200	NaN
feature embedding size	100	100	100	768
total parameters	575K	924K	402K	123M

F. Implementation details

We ran our experiments on Compute Canada Narval HPC [36] with Nvidia A100 GPUs and a Linux server with four Nvidia RTX 3090 GPUs, AMD Ryzen 48-Core CPU with 256 GB Ram. We used the implementations of studied DLVD approaches published in their GitHub repositories. For LIME and GNNExplainer, we used the implementation in Github-Lime and dgl-GNNExplainer [37].

We fine-tuned the DLVD approaches using the recommended parameters in the original papers. The parameter settings are presented in Table III. Noted that we used a learning rate of 5e-5 in the LineVul model because it is a pre-trained model and our training on the model can be considered as fine-tuning the Codebert for a downstream task – DLVD. Thus the learning rate is lower than other models. Our experiments involve 12 experimental instances, and for each

experimental instance we have 7 data sampling approaches (NoSampling plus various sampling approaches) to examine. Therefore, we have 84 (7*12) combinations. For each data sampling approach, we ran 20 times for getting reliable results. We totally ran 12x7x20 = 1,680 experiments, which costs more than 10,200 GPU hours.

IV. Results

A. RQ1: Does data sampling improve the effectiveness of existing DLVD approaches?

TABLE IV: Average rank of each data sampling approach in terms of different evaluation metrics. For each metric, we highlight the records that have a smaller rank than NoSampling with green, and darker color indicates better.

Sampling	AUC	P	R	F1	P@10	P@20	P@50	P@100
NoSampling	3.9	3.3	6.8	6.6	4.0	3.8	4.2	4.3
RUS_R	3.0	4.6	2.3	3.3	4.5	4.7	4.4	4.2
ROS_R	3.4	2.9	4.3	2.8	2.4	2.5	2.5	2.5
SMOTE_L	5.3	5.0	3.3	3.3	4.3	3.8	3.8	3.9
OSS_L	4.3	3.3	4.8	4.5	4.8	4.7	4.2	4.8
RUS_L	3.8	5.1	2.5	3.8	4.2	4.8	4.8	4.5
ROS_L	4.3	3.9	4.0	3.8	3.8	3.8	4.0	3.9

Table V presents the detailed results of various data sampling approaches in the 12 experimental instances in terms of various evaluation metrics. We observe that almost all data sampling techniques improve the effectiveness of all experimental instances in terms of F1, which is a major evaluation metric for the binary classification task. We observe the same results in Table IV which presents the average rank of each studied data sampling approach including the NoSampling across the 12 experimental instances, all studied data sampling approaches have a smaller rank than NoSampling in terms of F1. Similar results are observed when looking at P@k on imbalanced datasets (Reveal and BigVul). For instance, in terms of P@10, at least one data sampling approach outperforms NoSampling in all experimental instances except the instances related to the Reveal model.

> ***Finding1:*** *In general, data sampling improves the effectiveness of all studied models, especially on imbalanced datasets.*

In Table IV, only ROS_R outperforms NoSampling in terms of all evaluation metrics. In Table V, we observe that ROS_R wins (ranks 1st) 7 out of the 12 instances in terms of F1. If only considering the imbalanced datasets, ROS_R wins 7 out of 8 instances in terms of F1, except $Devign + Reveal$. ROS_R improves the NoSampling in terms of F1 across all experimental instances. Although ROS_R is not the best in $Devign + Reveal$, it still improves NoSampling by 31.9% from 0.188 to 0.248. When considering the improvement of ROS_R over NoSampling in the 8 experimental instances on imbalanced datasets, ROS improves the NoSampling at least by 31.9%. In terms of P@10, ROS_R outperforms NoSampling in 5 out of the 8 instances on imbalanced datasets. Surprisingly, ROS_R performs much better than SMOTE_L in all evaluation metrics, which is an advanced sampling approach and is reported at least comparable to random oversampling reported in prior studies [38]–[40].

> ***Finding2:*** *Surprisingly, the simple approach ROS_R outperforms all other studied sampling approaches (including the advanced one SMOTE_L) and it significantly improves the effectiveness of DLVD approaches in terms of all studied metrics.*

When comparing oversampling with undersampling, we observe that ROS_R outperforms RUS_R in all metrics except AUC and R in Table IV. Similarly, ROS_L outperforms RUS_L in all P@k and R. Such a finding is expected since undersampling deleting data points from the dataset, which causes information loss while oversampling keeps all the original data. Interestingly, when looking at the improvement in recall, we observe that RUS_R achieves the best performance in all experimental instances on imbalanced datasets (Reveal and BigVul), except for $LineVul + Reveal$, in which RUS_L performs the best. Although in some instances, RUS_R is not the best, RUS_R still improves the recall across all the experimental instances at least with an improvement of 10% over NoSampling. Our finding is compatible with prior studies [41] that undersampling achieves the largest improvement of recall. We investigate the reason behind this in Section V-B.

> ***Finding3****: In general, oversampling outperforms undersampling, while random undersampling performs the best in improving recall on imbalanced datasets.*

When comparing sampling_L with sampling_R, we observe that ROS_R outperforms ROS_L in terms of all metrics in Table IV. For RUS, RUS_R has a similar rank as RUS_L in terms of all P@k, however, RUS_R outperforms RUS_L in terms of AUC, P, R, and F1. Therefore, in general, sampling_R outperforms sampling_L, typically for ROS. We speculate the possible reasons in two folds. First, projecting the code into latent space leads to information loss and brings bias compared with the original code. Sampling on the latent space magnifies such bias. Second, since the sampling is applied after the representation learning, in other words, data sampling does not provide help in the representation learning process.

> ***Finding4****: In general, sampling_R outperforms sampling_L.*

Table 1 shows that for model Reveal and Devign which perform relatively well on the original datasets (e.g., Devign, Reveal, and BigVul), OSS helps improve their performance (e.g., in terms of F1). While OSS does not help DLVD approaches in handling data imbalance for the model IVDetect and LineVu which perform poorly on imbalanced datasets (i.e., Reveal and BigVul) in the NoSampling scenario. For instance, the IVDetect and LineVul perform poorly on imbalanced datasets (Reveal and BigVul), i.e., values of all metrics nearly equal to 0. The reason is that those two models do not have tolerance for the imbalance in training data and they predict all data points as non-vulnerable. Applying

TABLE V: The results of each data sampling approach in terms of different evaluation metrics, including NoSampling. To indicate which sampling approach is more effective than NoSampling in each metric, the records that have a larger value than the NoSampling are highlighted in green, and darker color indicates better. The best records for each experimental instance in terms of F1 are boldened.

	DLVD	IVDetect								Devign							
	Sampling	AUC	P	R	F1	P@10	P@20	P@50	P@100	AUC	P	R	F1	P@10	P@20	P@50	P@100
Devign	NoSampling	0.573	0.534	0.494	0.512	0.7	0.65	0.64	0.67	0.569	0.504	0.479	0.49	0.788	0.763	0.678	0.648
	RUS_R	0.596	0.524	0.523	0.547	0.6	0.6	0.62	0.65	0.564	0.501	0.532	0.514	0.62	0.59	0.648	0.608
	ROS_R	0.598	0.528	0.544	0.535	0.7	0.75	0.66	0.68	0.567	0.505	0.488	0.495	0.7	0.7	0.678	0.64
	SMOTE_L	0.555	0.483	0.858	**0.61**	0.667	0.631	0.626	0.618	0.583	0.511	0.557	0.527	0.74	0.73	0.68	0.686
	OSS_L	0.608	0.535	0.549	0.542	0.629	0.621	0.614	0.611	0.584	0.51	0.569	**0.535**	0.64	0.67	0.684	0.67
	RUS_L	0.608	0.537	0.538	0.537	0.676	0.645	0.615	0.62	0.584	0.512	0.534	0.517	0.74	0.66	0.652	0.656
	ROS_L	0.607	0.537	0.533	0.535	0.695	0.645	0.626	0.619	0.584	0.509	0.566	0.531	0.68	0.7	0.668	0.68
Reveal	NoSampling	0.402	0.008	1E-04	3E-04	0	0	0	0	0.711	0.274	0.188	0.188	0.838	0.775	0.625	0.541
	RUS_R	0.718	0.196	0.688	0.305	0.4	0.3	0.3	0.31	0.701	0.178	0.656	0.278	0.4	0.3	0.325	0.33
	ROS_R	0.71	0.286	0.492	**0.46**	0.6	0.45	0.44	0.43	0.697	0.213	0.3	0.248	0.867	0.839	0.813	0.786
	SMOTE_L	0.624	0.142	0.593	0.228	0.265	0.303	0.242	0.22	0.695	0.248	0.397	0.304	0.914	0.886	0.814	0.75
	OSS_L	0.38	0	0	0	0	0	0	0	0.706	0.273	0.287	0.279	0.943	0.907	0.84	0.741
	RUS_L	0.619	0.134	0.661	0.22	0.26	0.305	0.245	0.213	0.708	0.25	0.459	**0.322**	0.886	0.836	0.789	0.734
	ROS_L	0.622	0.139	0.626	0.226	0.275	0.303	0.237	0.22	0.697	0.25	0.414	0.31	0.943	0.921	0.834	0.733
BigVul	NoSampling	0.461	0	0	0	0	0	0	0	0.701	0.326	0.201	0.247	0.68	0.72	0.668	0.586
	RUS_R	0.679	0.111	0.628	0.188	0.2	0.2	0.14	0.12	0.687	0.1	0.682	0.174	0.2	0.18	0.156	0.146
	ROS_R	0.77	0.244	0.622	**0.347**	0.2	0.1	0.14	0.15	0.703	0.295	0.253	**0.271**	0.48	0.42	0.472	0.488
	SMOTE_L	0.568	0.08	0.507	0.137	0.06	0.088	0.08	0.097	0.717	0.259	0.284	0.269	0.614	0.65	0.631	0.557
	OSS_L	0.447	0.1	5E-05	1E-04	0.005	0.003	0.001	5E-04	0.712	0.316	0.221	0.253	0.714	0.686	0.611	0.577
	RUS_L	0.561	0.075	0.584	0.131	0.065	0.088	0.086	0.094	0.711	0.221	0.354	0.268	0.714	0.714	0.646	0.571
	ROS_L	0.566	0.078	0.541	0.136	0.07	0.088	0.084	0.097	0.723	0.232	0.333	0.27	0.7	0.714	0.654	0.6
	DLDV	**Reveal**								**LineVul**							
	Sampling	**AUC**	**P**	**R**	**F1**	**P@10**	**P@20**	**P@50**	**P@100**	**AUC**	**P**	**R**	**F1**	**P@10**	**P@20**	**P@50**	**P@100**
Devign	NoSampling	0.555	0.512	0.42	0.46	0.525	0.5	0.438	0.444	0.71	0.616	0.537	0.573	0.942	0.942	0.934	0.918
	RUS_R	0.553	0.499	0.545	**0.519**	0.47	0.47	0.446	0.46	0.704	0.595	0.59	**0.592**	0.989	0.981	0.973	0.967
	ROS_R	0.553	0.508	0.492	0.497	0.49	0.445	0.416	0.418	0.705	0.606	0.557	0.58	0.968	0.971	0.973	0.965
	SMOTE_L	0.54	0.495	0.511	0.502	0.42	0.458	0.474	0.467	0.661	0.563	0.618	0.586	0.95	0.95	0.937	0.919
	OSS_L	0.54	0.494	0.547	**0.519**	0.44	0.45	0.468	0.458	0.663	0.564	0.587	0.573	0.95	0.945	0.937	0.912
	RUS_L	0.541	0.493	0.525	0.508	0.405	0.435	0.436	0.44	0.661	0.56	0.616	0.585	0.965	0.943	0.932	0.916
	ROS_L	0.541	0.496	0.506	0.5	0.415	0.445	0.442	0.453	0.66	0.568	0.554	0.558	0.965	0.953	0.929	0.912
Reveal	NoSampling	0.583	0.282	0.119	0.159	0.625	0.556	0.56	0.484	0.775	0.4	0.249	0.299	0.73	0.72	0.683	0.625
	RUS_R	0.674	0.178	0.594	0.273	0.425	0.45	0.415	0.395	0.841	0.74	0.284	0.41	0.512	0.538	0.508	0.491
	ROS_R	0.636	0.23	0.414	**0.292**	0.61	0.61	0.588	0.576	0.811	0.443	0.506	**0.472**	0.965	0.963	0.948	0.907
	SMOTE_L	0.621	0.192	0.311	0.235	0.415	0.343	0.237	0.215	0.716	0.301	0.51	0.355	0.735	0.74	0.687	0.625
	OSS_L	0.636	0.245	0.211	0.219	0.34	0.318	0.255	0.244	0.72	0.421	0.248	0.295	0.72	0.742	0.687	0.626
	RUS_L	0.676	0.187	0.557	0.277	0.31	0.325	0.34	0.3	0.724	0.263	0.599	0.354	0.755	0.715	0.679	0.646
	ROS_L	0.63	0.197	0.311	0.24	0.29	0.265	0.239	0.203	0.724	0.374	0.352	0.357	0.745	0.728	0.701	0.647
BigVul	NoSampling	0.679	0.387	0.109	0.168	0.72	0.7	0.648	0.578	0.554	0.07	0.002	0.005	0.07	0.07	0.063	0.046
	RUS_R	0.684	0.107	0.617	0.182	0.28	0.23	0.248	0.242	0.825	0.164	0.722	0.266	0.59	0.618	0.627	0.592
	ROS_R	0.661	0.232	0.304	**0.261**	0.457	0.5	0.477	0.444	0.745	0.455	0.367	**0.406**	0.947	0.958	0.943	0.904
	SMOTE_L	0.617	0.19	0.276	0.224	0.34	0.375	0.394	0.398	0.532	0.062	0.587	0.112	0.175	0.145	0.133	0.127
	OSS_L	0.632	0.248	0.224	0.235	0.41	0.43	0.437	0.44	0.544	0.064	0.003	0.005	0.075	0.068	0.056	0.043
	RUS_L	0.664	0.114	0.564	0.189	0.29	0.29	0.273	0.279	0.542	0.07	0.559	0.11	0.175	0.15	0.14	0.142
	ROS_L	0.62	0.193	0.285	0.23	0.335	0.323	0.338	0.353	0.539	0.075	0.501	0.121	0.18	0.163	0.142	0.147

OSS_L does not help improve the effectiveness of IVDetect and LineVul. Why can OSS not balance data well? OSS relies on Tomek Links [24] and Condensed Nearest Neighbor (CNN) [42] to decide which non-vulnerable data to delete. However, such deleted non-vulnerable data points have to meet certain criteria. OSS will stop deleting non-vulnerable data if no more data points are found to meet the criteria even the data still remains imbalanced after running OSS. Therefore, OSS does not guarantee to reduce the size of the majority class to the minority class. We investigate the datasets Reveal and BigVul after applying OSS_L and find that the datasets are still extremely imbalanced. The ratio of vulnerable and non-vulnerable is 1:8.7 for the Reveal dataset, and 1:16 for the BigVul dataset.

Finding5: *Surprisingly, OSS does not help alleviate the data imbalance problem in DLVD if the DLVD approaches (i.e., Reveal and BigVul) perform poorly on imbalanced data originally.*

B. RQ2: Does data sampling improve the ability of DLVD for learning the vulnerable patterns?

TABLE VI: The hit ratio of various data sampling approaches in the experimental instances $LineVul + BigVul$ and $Reveal + BigVul$.

	$LineVul + BigVul$				$Reveal + BigVul$			
	Top1	Top3	Top5	Top10	Top1	Top3	Top5	Top10
NoSampling	0.39	0.55	0.66	0.66	0.55	0.76	0.81	0.90
ROS_R	0.41	0.62	0.72	0.82	0.67	0.83	0.89	0.92
RUS_R	0.32	0.45	0.56	0.71	0.69	0.85	0.91	0.95
ROS_L	0.22	0.39	0.52	0.68	0.65	0.84	0.90	0.95
RUS_L	0.15	0.37	0.50	0.66	0.65	0.86	0.91	0.95
SMOTE_L	0.13	0.35	0.47	0.63	0.67	0.84	0.90	0.95
OSS_L	0.32	0.53	0.61	0.72	0.69	0.84	0.90	0.95
ROS_R_2X	0.47	0.67	0.76	0.85	0.67	0.83	0.89	0.93
ROS_R_4X	0.45	0.66	0.75	0.85	0.64	0.86	0.91	0.94

The hit ratio of various data sampling approaches in $LineVul + BigVul$ and $Reveal + BigVul$ are shown in TableVI. In $Reveal + BigVul$, all studied data sampling

approaches improve hit ratio compared with NoSampling. In $LineVul + BigVul$, only ROS_R improves the hit ratio. A possible explanation is that LineVul does not learn well on BigVul on imbalanced data, and only ROS_R helps handle data imbalance (see the effectiveness of LineVul on BigVul in Table V). Nevertheless, ROS_R improves the NoSampling on both two instances. For instance, after ROS_R, the hit ratio in $LineVul + BigVul$ increases from 0.39, 0.55, 0.66, and 0.66 to 0.41 0.62, 0.72, and 0.82 when k equals to 1, 3, 5, and 10 with improvements of 5.6%, 12.7%, 9.1%, and 24.2%, respectively. Similarly, ROS_R improves the hit ratio in $Reveal + BigVul$ by 21.8%, 8.6%, 9.4%, and 2.4% when k equals to 1, 3, 5, and 10, respectively. Such results suggest that ROS_R helps LineVul and Reveal learn better about the characteristics of vulnerable patterns. For example, Figure 2 presents the LIME result of a vulnerable function before and after ROS_R. The function has a Use-after-free vulnerability at lines marked with green rectangle [43]. Before ROS_R, LineVul predicts it as non-vulnerable according to the statement highlighted in blue, while after ROS_R, LineVul is able to predict correctly and really based on the vulnerable lines highlighted in orange. The observation probably indicates that **duplicated data points in the raw training dataset are not necessarily noisy to the model, they could provide a positive influence on the model for learning the real patterns for detecting vulnerabilities when added properly.** To validate this assumption, we increase the ratio of vulnerable and non-vulnerable cases ($Ratio_{Vul}$) from 1:1 to 2:1, and 4:1 using ROS_R and compare their hit ratio. The results are presented in Table VI for different $Ratio_{Vul}$, i.e., ROS_R for 1:1, ROS_R_2X for 2:1 and ROS_R_4X for 4:1. We can see that the hit ratio for top 1 increases from 0.41 to 0.47 when $Ratio_{Vul}$ increases from 1 to 2, and slightly drops when the $Ratio_{Vul}$ reaches 4 (0.45) for LineVul. Meanwhile the effectiveness of LineVul and Reveal increases after increasing the repetition as well, i.e., F1 values for LineVul are 0.406, 0.42, and 0.42 after applying ROS_R, ROS_R_2X, and ROS_R_4X, respectively. We observe a similar phenomenon for the Reveal.

We further perform additional analysis to understand the impact of ROS_R with different multiplications (i.e., 2x and 4x) of vulnerable cases on model LineVul for dataset BigVul (LineVul+BigVul) in terms of effectiveness. The results show that the F1 scores are 0.406 (ROS_R) to 0.420 (ROS_R_2x), and 0.418 (ROS_R_4x), respectively. We observe the same trend for AUC. The Top1 hit ratios arrive the best value at 2x and slightly drop afterward. Too many multiplications of vulnerable cases could bias the classification and decrease the effectiveness of the model. The results suggest that ROS_R_2x works best for the LineVul+BigVul instance. The result indicates that increasing the size of repeated vulnerable cases properly could both improve the effectiveness and its ability of learning vulnerable patterns.

Finding6: *ROS_R improves the ability of DLVD approaches for learning real vulnerable patterns.*

Although our results show that ROS_R can improve the ability of DLDV for capturing vulnerable patterns and predicting based on them. It still has potential for future improvement even after applying data sampling. If we look at the top1 results, the hit ratios are 0.39 and 0.55 for $LineVul+BigVul$ and $Reveal + BigVul$, respectively. Even after applying RUS_R, hit ratios are improved to 0.42 and 0.67. There are still a significant number of cases in which the decision is not really made based on the vulnerable statements. 58% and 33% of the vulnerable functions that are predicted as vulnerable are not based on their real vulnerable statements for LineVul and Reveal.

Finding7: *In a significant portion of cases (at least 33%), DVLD approach cannot reason their prediction over real vulnerable statements. There is still room for improving the ability of learning real vulnerable patterns for DLVD approaches.*

V. Discussion

A. Why does ROS_R perform the best among all studied approaches?

In RQ1, we observe that ROS_R performs the best among all studied approaches in all studied metrics across experimental instances, typically for imbalanced datasets. Why is RQS_R the king? To some extent, the results in RQ2 provide insights, i.e., ROS_R improves the ability of DLVD approaches for learning real vulnerable patterns. We speculate ROS_R performs the best due to its natural compared with other approaches. First, compared with undersampling (i.e., RUS_R), instead of deleting non-vulnerable code from the training data which causes information loss, ROS_R, as an oversampling approach does not have this issue. Second, compared with other sampling approaches on latent space, ROS_R performs sampling on raw data. We speculate that sampling_R separates vulnerable cases from non-vulnerable cases better than sampling_L. For example, to illustrate the effect of ROS_L and ROS_R on the data distribution in latent space, we used the t-distributed stochastic neighbor embedding (t-SNE) [44] to visualize the distribution of the data points when using ROS_L and using ROS_R in $LineVul+BigVul$, in which the F1 value for ROS_R and ROS_L are 0.406 and 0.121, respectively. Figure 3 presents the distribution of data points after applying ROS_R and ROS_L. Vulnerable and non-vulnerable data is better separated when applying ROS_R than applying ROS_L.

B. Why does random undersampling boost recall?

Recall measures how well a DLVD approach could identify all vulnerable code. Prior studies report that there is a significant portion of similar code in code corpus [45]–[47]. Therefore, we assume there probably exists a significant number of similar non-vulnerable cases surrounding vulnerable cases and introduce significant noise for models to identify various vulnerable code. Random undersampling reduces such noise,

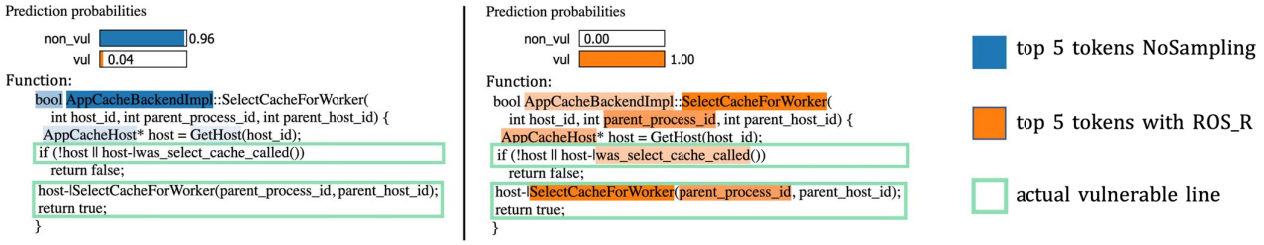

Fig. 2: The explanation of LIME on a vulnerable function, which is predicted wrongly as non-vulnerable before ROS (left) and predicted correctly after ROS_R (right).

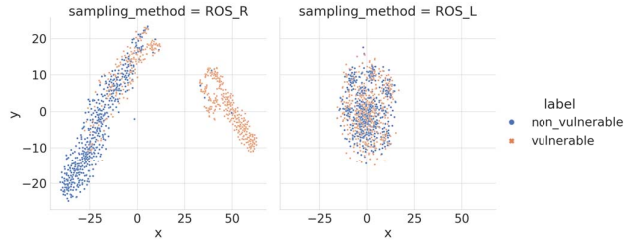

Fig. 3: The distribution of data points in the experiment instance $LineVul+BigVul$ after applying ROS_R and ROS_L.

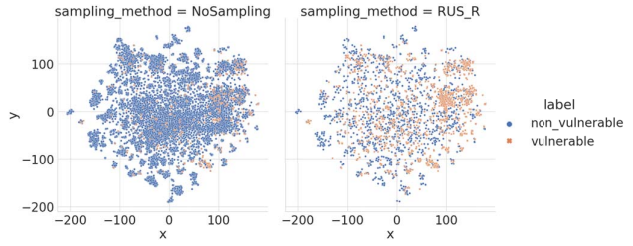

Fig. 4: The distribution of data points in the Reveal dataset without (NoSampling) and without applying RUS_R.

therefore boosts recall, although it reduces precision due to information loss.

To validate our assumption, we conduct a case study on the experimental instances for the Reveal model. We applied t-SNE on the representation vectors generated by the Reveal model and visualize the distribution of points before and after RUS_R which performs the best. To quantitatively validate our assumption, we also calculated the mean and medium number of neighbors for each vulnerable point in the generated t-SNE plot. We consider all non-vulnerable points having a Euclidean distance less than 4 to a vulnerable point as its neighbor, given the map size is 400*400. We discuss the threats to validity caused by the threshold selection (i.e., 4) in Section V-D.

Figure 4 presents data points distribution for $Reveal + Reveal$ (due to the space limitation we do not show the plots for another two datasets). We observe that without applying RUS_R (NoSampling), non-vulnerable points cover the majority of non-vulnerable points, and vulnerable and non-vulnerable are almost mixed together. After applying RUS_R, better separation between the vulnerable and non-vulnerable points is observed. Table VII presents the median/mean number of neighbors for each vulnerable point before and after RUS_R. On all studied datasets, the mean/median number is reduced significantly, but the ratio of reduction varies in different datasets. The mean is reduced from 6.68 to 5.43 on balanced data (Devign), while on the imbalanced datasets, the mean is reduced from 38.11 to 3.6 and 41.43 to 2.6 on the Reveal and BigVul datasets with a reduction of more than 10 times. We observer a correlation between the improvement of recall and the size of the reduction. The recall achieves an improvement of 464% and an improvement of 397% on BigVul and Reveal datasets, respectively, while only obtains an improvement of 12.4% on Devign. Such results explain to some extent why the random undersampling can reduce noise for vulnerable code and thus boost recall.

TABLE VII: Number of neighbors before and after RUS_R in the experimental instances related to the Reveal model.

Dataset	**Devign**		**Reveal**		**BigVul**	
Sampling	mean	median	mean	median	mean	median
NoSampling	6.68	6	38.11	38	41.43	43
RUS_R	5.43	5	3.60	3	2.60	2

C. Implications of our findings

We recommend future practitioners to use oversampling over undersampling, sampling_R over sampling_L, and typically to use ROS_R to handle data imbalance issue in DLVD. In RQ1, we observe that generally oversampling outperforms undersampling, sampling_R outperforms sampling_L. Typically, ROS_R performs the best among all studied sampling approaches. Moreover, we observe that ROS_R can improve the ability of learning real vulnerable patterns in source code for DLVD approaches (both text-based and graph-based models). Therefore, ROS_R is recommended for handling data imbalance issue in DLVD.

We do not recommend practitioners to use OSS_L for imbalanced data in DLVD. In RQ1, we observe that among all studied sampling approaches, OSS_L is the only one that does not help improve the effectiveness of DLVD on imbalanced datasets at all due to its mechanism that cannot guarantee to really balance the target dataset, typically when the ratio between vulnerable and non-vulnerable code is large.

Therefore, we do not suggest practitioners to use OSS_L on imbalanced data for training DLVD.n

We recommend future practitioners to use RUS_R if they wish to improve the recall. In RQ1, we observe that undersampling approaches, typically RUS_R, can boost the recall for DLVD approaches. In Section V-B, our analysis shows that RUS_R reduces the number of surrounding similar non-vulnerable points for vulnerable points dramatically to reduce noise. In practice, if practitioners aim to improve the recall of their approaches, they probably could consider using RUS_R.

We encourage future research to develop new data augmentation techniques to improve the ability of DLVD approaches for learning real vulnerable patterns. In RQ2, although our results show that ROS_R can improve the ability of DLDV for learning vulnerable patterns. However, there are still a significant number of cases (58% and 33% of the vulnerable functions in $LineVul + BigVul$ and $Reveal + BigVul$), in which the decision is not made based on the vulnerable statements. It still has potential for future improvement even after applying data sampling. Our findings shed light for addressing/alleviating this problem. For instance, data augmentation probably is a valuable direction to investigate, as our results show that the simple repetition strategy (e.g., ROS_R) is helpful in RQ2.

D. Threats to validity

Internal Validity One threat relates to hyperparameter settings when training the studied DLVD approaches. As hyperparameter tuning is extremely expensive for our studied approaches that consist of millions of parameters. We used the recommended parameters from previous studies. In addition, time-wise evaluation scenarios (use the earlier data for training and later data for testing) are not considered in this paper, since the dataset is at the function level, not the commit level. Note that the results obtained for the studied approaches (IVDetect, Devign, Reveal, and LineVul) are not exactly the same as the results reported in the original papers. One possible explanation is that we ran the experiment 20 times and take an average, while they report the best one. To migrate the threat, we reused the implementations published by the authors and followed the experimental setup of DLVD approaches specified in their papers. Nevertheless, our goal is not to compare the effectiveness of those DLVD approaches, instead we focus on studying the impact of data sampling approaches on the DLVD approaches. In Section V-B, we selected 4 as the threshold to calculate the neighbourhood of vulnerable cases, which may introduce bias to the results. To migrate the threat, we performed the same experiments using different thresholds and our finding still hold. Similarly, in RQ2, we selected to examine the k tokens returned by LIME and GNNExplainer, different k may lead to different results. To migrate this threat, we examined different values 1, 3, 5, 10 for k and find the finding is hold for different k. In addition, recent study LIME can be too random and may generate inconsistent results of important tokens [48] and may introduce bias to our study. We encourage future studies to try more advanced techniques for the explanation. We published our replication package to improve the transparency of our work.

External validity Threats to the external validity relate to the generalizability of our findings. Our findings might not be generalized to other datasets, DLVD approaches, data sampling approaches. We encourage future research on more approaches and datasets.

VI. Related work

A. Machine learning-based Vulnerability detection

With the rapid development of machine learning and its successful application in various fields, typically deep learning. Researchers have started to investigate machine learning and deep learning-based vulnerability detection methods. Early research of machine learning-based vulnerability detection approaches typically requires human-defined features to identify vulnerability and then use machine learning models to train on these features [1], [49]. For example, Scandariato et al. [1] use text mining technology and leverage the frequency of occurrence of specific terms for vulnerability detection. Unlike machine learning approaches, deep learning-based methods usually do not define features in advance, but let the model extract features for learning by itself through a deeper network. See Section II-A for more details on the prior DLVD approaches. Napier et al. found that text-based machine learning models are not effective in detecting vulnerabilities within or across projects and vulnerability types [50]. Different from prior studies which focus on improving the effectiveness of ML-based VD approaches, we investigated the impact of various data sampling approaches on DLVD approaches on an extensive scale. We also investigate the reason behind our findings and provide actionable suggestions for future practitioners and researchers.

B. Data sampling for software engineering tasks

Data sampling have been used for handling data imbalanced issue in various software engineering tasks, such as defect prediction [6], [7], [38], [40], [51]–[55], bug classification [54], [56], software quality prediction [8], and software change prediction [9]. Most studies show data sampling help improve of the given tasks [6], [7], [53]–[55], [57], which similar to our findings. For instance, Yedida and Menzies examined the value of oversampling for deep learning in defect prediction using deep learning [55] and show that oversampling (fuzz sampling) can significantly improve the prior SOTA DL approaches in the majority defect dataset. Similar finding is observed in our study, oversampling can improve SOTA DLVD. Zheng et al. conducted a comparative Study of class rebalancing methods for security bug report classification [54]. They evaluated various sampling approaches (e.g., SMOTE, ADASYN, and Rose) on multiple classifiers (e.g., logistic regression and random forest) and found that the combination of Rose + random forest performs the best. Kamei et al. compare undersampling and oversampling in the task of fault-prone module detection and found that undersampling and oversampling

have similar effectiveness [57], while our findings suggest that oversampling is better than undersampling in DLVD. More important, different from prior studies, we focus on a new task – DLVD. We also compare two sampling strategies - latent space sampling vs. raw data sampling, and study whether data sampling could improve the ability of DLVD for capturing real vulnerable patterns.

VII. Conclusion

We conducted the first systematic and extensive study to assess the impact of data sampling on data imbalance issue in SOTA DLVD approaches. Generally, oversampling outperforms undersampling, and sampling on raw data outperforms sampling on latent space, typically random oversampling on raw data performs the best among all studied ones (including advanced one SMOTE and OSS). Surprisingly, OSS does not help alleviate the data imbalance issue in DLVD at all. While if recall is pursued, random undersampling is the best choice. Random oversampling on raw data also improves the ability of DLVD approaches for learning real vulnerable patterns. However, for a significant portion of cases (at least 33% in our datasets), DLVD approaches cannot reason their prediction based on real vulnerable statements. We provide actionable suggestions and a roadmap to practitioners and researchers, e.g., random oversampling is recommended to handle data imbalance issue in DLVD while OSS is not recommended.

References

[1] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014.

[2] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.

[3] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Transactions on Software Engineering*, 2021.

[4] D. Hin, A. Kan, H. Chen, and M. A. Babar, "Linevd: Statement-level vulnerability detection using graph neural networks," in *IEEE/ACM 19th International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*, 2022, pp. 596–607.

[5] Y. Li, S. Wang, and T. N. Nguyen, "Vulnerability detection with fine-grained interpretations," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 292–303.

[6] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE transactions on reliability*, vol. 61, no. 2, pp. 516–525, 2012.

[7] H. Xu, R. Duan, S. Yang, and L. Guo, "An empirical study on data sampling for just-in-time defect prediction," in *International Conference on Artificial Intelligence and Security*. Springer, 2021, pp. 54–69.

[8] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 6, pp. 1283–1294, 2009.

[9] R. Malhotra and M. Khanna, "An empirical study for software change prediction using imbalanced data," *Empirical Software Engineering*, vol. 22, no. 6, pp. 2806–2851, 2017.

[10] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *CoRR*, vol. abs/1106.1813, 2011. [Online]. Available: http://arxiv.org/abs/1106.1813

[11] M. Kubat, S. Matwin *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," in *Icml*, vol. 97, no. 1. Citeseer, 1997, p. 179.

[12] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[13] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," 2022.

[14] M. T. Ribeiro, S. Singh, and C. Guestrin, """ why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[15] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR*, vol. 2013, 01 2013.

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[18] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[19] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[20] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[21] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 590–604.

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

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[24] I. Tomek, "Two Modifications of CNN," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7(2), pp. 679–772, 1976.

[25] C. X. Ling and C. Li, "Data mining for direct marketing: Problems and solutions." in *Kdd*, vol. 98, 1998, pp. 73–79.

[26] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "Ac/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 508–512.

[27] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.

[28] G. K. Rajbahadur, S. Wang, G. Ansaldi, Y. Kamei, and A. E. Hassan, "The impact of feature importance methods on the interpretation of defect classifiers," *IEEE Transactions on Software Engineering*, 2021.

[29] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering," *IEEE Transactions on Software Engineering*, vol. 47, no. 7, pp. 1414–1430, 2019.

[30] T.-D. B. Le, D. Lo, and M. Li, "Constrained feature selection for localizing faults," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 501–505.

[31] A. Okutan and O. T. Yıldız, "Software defect prediction using bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.

[32] N. Chen, S. C. Hoi, and X. Xiao, "Software process evaluation: A machine learning approach," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 333–342.

[33] W. Fu, V. Nair, and T. Menzies, "Why is differential evolution better than grid search for tuning defect predictors?" *arXiv preprint arXiv:1609.02613*, 2016.

[34] J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, and A. E. Hassan, "Finding a needle in a haystack: Automated mining of silent

vulnerability fixes," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 705–716.
[35] D. Zou, Y. Zhu, S. Xu, Z. Li, H. Jin, and H. Ye, "Interpreting deep learning-based vulnerability detector predictions based on heuristic searching," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–31, 2021.
[36] https://docs.alliancecan.ca/wiki/Narval/en/.
[37] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.
[38] K. E. Bennin, J. Keung, A. Monden, P. Phannachitta, and S. Mensah, "The significant effects of data sampling approaches on software defect prioritization and classification," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 364–373.
[39] L. Gong, S. Jiang, and L. Jiang, "Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering," *IEEE Access*, vol. 7, pp. 145 725–145 737, 2019.
[40] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Transactions on Software Engineering*, vol. 44, no. 6, pp. 534–550, 2017.
[41] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200–1219, 2018.
[42] P. Hart, "The condensed nearest neighbor rule (corresp.)," *IEEE transactions on information theory*, vol. 14, no. 3, pp. 515–516, 1968.
[43] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-6766.
[44] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.
[45] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 437–440.
[46] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1936–1964, 2017.
[47] M. Gharehyazie, B. Ray, and V. Filkov, "Some from here, some from there: Cross-project code reuse in github," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 291–301.
[48] C. Pornprasit, C. Tantithamthavorn, J. Jiarpakdee, M. Fu, and P. Thongtanunam, "Pyexplainer: Explaining the predictions of just-in-time defect models," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 407–418.
[49] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, "Graph-based mining of multiple object usage patterns," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 383–392. [Online]. Available: https://doi.org/10.1145/1595696.1595767
[50] N. Kollin, B. Tanmay, and W. Shaowei, "An empirical study of text-based machine learning models for vulnerability detection," *Empirical Software Engineering*, 2022.
[51] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 99–108.
[52] K. E. Bennin, J. W. Keung, and A. Monden, "On the relative value of data resampling approaches for software defect prediction," *Empirical Software Engineering*, vol. 24, no. 2, pp. 602–636, 2019.
[53] S. Feng, J. Keung, X. Yu, Y. Xiao, K. E. Bennin, M. A. Kabir, and M. Zhang, "Coste: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, vol. 129, p. 106432, 2021.
[54] W. Zheng, Y. Xun, X. Wu, Z. Deng, X. Chen, and Y. Sui, "A comparative study of class rebalancing methods for security bug report classification," *IEEE Transactions on Reliability*, vol. 70, no. 4, pp. 1658–1670, 2021.
[55] R. Yedida and T. Menzies, "On the value of oversampling for deep learning in software defect prediction," *IEEE Transactions on Software Engineering*, 2021.
[56] R. Shu, T. Xia, J. Chen, L. Williams, and T. Menzies, "How to better distinguish security bug reports (using dual hyperparameter optimization)," *Empirical Software Engineering*, vol. 26, no. 3, pp. 1–37, 2021.
[57] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *First international symposium on empirical software engineering and measurement (ESEM 2007)*. IEEE, 2007, pp. 196–204.