A Mutation-Based Data Enhancement Approach for Software Vulnerability Detection

*Lianyi Su, Jie Hu, Wei Zheng**

School of Software,
Northwestern Polytechnical University,
China
sulianyi@mail.nwpu.edu.cn, 305860691@qq.com, wzheng@nwpu.edu.cn

Abstract

Effective software vulnerability detection is paramount for ensuring the security of software systems. However, the presence of imbalanced data in extensive datasets often leads to overfitting on non-vulnerable code and suboptimal performance on vulnerable code. Traditional methods of collecting vulnerable data frequently fall short in capturing the complexities of real-world scenarios. This paper proposes a mutation-based data enhancement approach to tackle this challenge, with a focus on capturing essential traits of vulnerable source code. Our approach systematically extracts traits from extensive vulnerable source code and employs mutation operators to introduce high-level alterations. We evaluate the convergence of multiple mutation rounds using a diversity index to ensure consistent enhancements. By selecting the most effective mutation operators for subsequent model training, our approach achieves substantial accuracy improvements across diverse datasets and deep neural network models. This work represents the initial version of our approach, with continuous refinements underway to facilitate practical implementation and address real-world security challenges.

Keywords: Deep learning, Vulnerability detection, Data enhancement, Code mutation

1 Introduction

Software vulnerability refers to defects in the specific implementation of software or system policies and is the root cause of the prevalence of network attacks [1-2]. As shown in Figure 1, in the CVE [3] (Common Vulnerabilities and Exposures) report, the most authoritative vulnerability repository in the world, only 894 vulnerabilities were exposed in 1999; by the end of 2022, the total number of vulnerabilities reached 166462.

With the vigorous development of artificial intelligence (e.g., large language model [4], GPT-4 [5]), the deep integration of static detection with machine learning [6], and deep learning has become a research hotspot in academia. Researchers try to explore new intelligent software vulnerability mining methods for problems of vulnerability mining. Using deep learning technology to detect potential vulnerabilities and defects in software has become a feasible solution for the intelligent detection of software vulnerabilities [7]. Vulnerability detection falls into two categories: source code-based and binary program-based. However, real projects often suffer from limited vulnerability data and diverse vulnerability characteristics [8]. This can lead to accuracy issues when transitioning to actual projects. The scarcity of real-world vulnerability datasets and the demand for large data in deep learning pose challenges to intelligent software vulnerability detection.

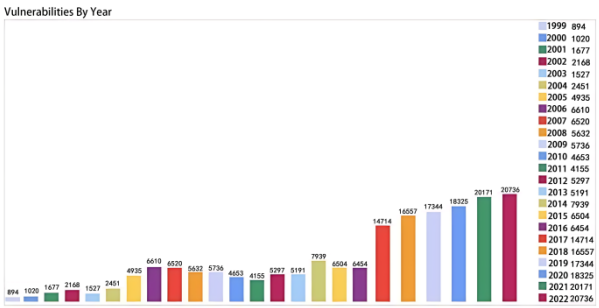

Figure 1. Number of vulnerabilities from 1999 to 2022 in CVE

This paper analyzes the current state of intelligent vulnerability detection and explores enhancing vulnerability code data as a means to improve detection accuracy. Data enhancement involves augmenting data by adding modified copies or synthetic data, which reduces overfitting during model training [8]. While widely used in computer vision and natural language processing, this paper adapts data enhancement techniques to the realm of vulnerable code.

Our contributions include proposing a code mutation-based data enhancement method [9-11], designing a mutation operator selection approach based on diversity, and evaluating the method across multiple open-source datasets and various deep-learning vulnerability detection models [12]. The paper proceeds as follows: Section 2 introduces background and related work, Section 3 presents our approach's algorithm, Section 4 details our experimental setup and results analysis, and Section 6 concludes the paper.

*Corresponding Author: Wei Zheng; E-mail: wzheng@nwpu.edu.cn
DOI: https://doi.org/10.70003/160792642024112506013

2 Background and Related Work

2.1 Definition and Characteristics of Software Vulnerabilities

Software vulnerabilities are security flaws in software or information systems that can allow unauthorized access and compromise system integrity [13-14]. They pose significant threats at various levels, necessitating effective detection. Exposure in spatial information security includes diversity [15], persistence, timeliness, concealment, and usability. Understanding these vulnerability characteristics, especially the information in the code, is crucial for efficient vulnerability detection. Enhancing vulnerability-related code texts through data augmentation is vital for improved detection [16].

2.2 Vulnerability Detection Neural Network

Recent advancements have yielded numerous effective deep neural network structures, enabling the application of deep learning in code vulnerability and defect prediction [17-18]. This presents exciting opportunities for in-depth research in software reliability. Figure 2 illustrates the structure of a simple neural network, comprising input and output layers, with intermediate layers referred to as hidden layers. An activation function is employed to introduce nonlinearity after weighing and summing the input vectors, enhancing the network's modeling capabilities. During training, the output layer's feature representation is typically processed by the softmax function to derive a posterior probability distribution [19-20]. The cross-entropy between this distribution and the empirical distribution serves as the loss function.

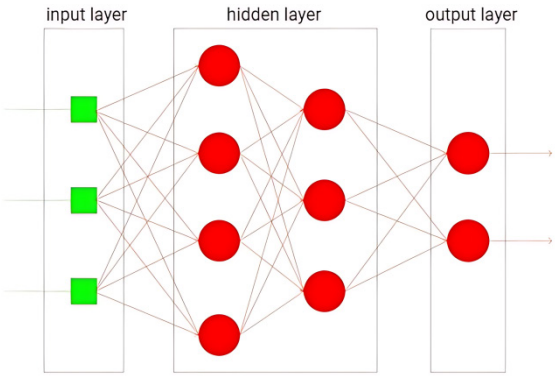

Figure 2. Simple neural network structure diagram

3 Our Data Enhancement Approach

This chapter mainly provides a detailed introduction to the data augmentation method studied in this article, as shown in Figure 3.

3.1 Vulnerability-Related Code Examples

Figure 4 shows a simple vulnerability code snippet. There is a memory out-of-bounds access vulnerability (CWE-119), and the code related to the exposure is marked in yellow. As shown in Figure 4, the function's function is to re-encode according to the input and copy it to the target buffer. The loophole of this function is that a character string larger than MAXSIZE can be obtained by constructing a line containing too many '&' symbols. Large-length yarns create illegal overwrites of memory. Analyzing the vulnerable code in the previous paragraph, arrays need to be defined as a standard implementation method of unlawful memory access type vulnerabilities because, under normal circumstances, we access illegal memory locations in the form of array out-of-bounds. In addition, a statement is needed to access the memory, so there will be an array of subscript index symbols and functions such as strcpy and memcpy; simultaneously, because it is an out-of-bounds access, there is no if statement in the code to check the subscript variable. It can be seen that in addition to the characteristics of the code itself, such as malloc, strcpy and other library functions, more in-depth features are needed to learn the process from "definition" to "no check" and then to "illegal access".

```
char * copy_input(char * user_supplied_string) {
  int i,  dst_index;

  char * dst_buf = (char*)malloc(4*sizeof(char)*MAX_SIZE);

  if (MAX_SIZE <= strlen(user_supplied_string)){
    LOG_FAIL("user string too long, die evil hacker!");
  }

  dst_index = 0;
  for(i= 0;i < strlen(user_supplied_string);i++) {

    if('&' == user_supplied_string[i]) {
      dst_buf[dst_index++] = '&';
      dst_buf[dst_index++] = 'a';
      dst_buf[dst_index++] = 'm';
      dst_buf[dst_index++] = 'p';
      dst_buf[dst_index++] = ';';
    } else if ('<' == user_supplied_string[i]) {
      /* encode to &lt; */
    } else {
      dst_buf[dst_index++]=user_supplied_string[i];
    }
  }

  return dst_buf;
}
```

Figure 3. Vulnerability code cases

Based on the above analysis, to preserve the characteristics of these vulnerabilities, an essential principle of the data enhancement method proposed in this paper is that the enhanced samples still maintain the features of the same vulnerabilities based on as diverse representations as possible. From the subjective analysis of the code in Figure 4, you can start with functions, statements, control structures, and variable names and do as many "synonymous substitutions" as possible. To sum up, the research of this paper mainly focuses on two points: one is to do data enhancement without changing the type and characteristics of the vulnerability; the second approach involves enhancing the exposure of vulnerability code by leveraging data flow similarities. This enables the detection model to better understand vulnerable code during representation.

3.2 Data Enhancement Based on Code Mutation

Mutation testing involves modifying a program in small ways [21]. However, the same strategy can also be applied

to data. To improve the vulnerability dataset effectively, we utilize high-order mutation, a method that combines three mutation operators creatively to maximize data diversity through random permutations. The effectiveness of a mutation path is evaluated using the Self-BLEU index.

In our endeavor for an effective mutation path, proper code formatting is essential to align with the exact match statement for mutation operators. This process involves filtering out comments and blank lines, eliminating consecutive blank lines, and standardizing code indentation to four spaces, enhancing the identification of variation points and ranges. The code formatting process is illustrated in Figure 5. Additionally, we address duplicate data concerns using Simhash text de-duplication during formatting, which reduces dimensionality and calculates text similarity by evaluating hamming distance between hash strings. The specific process is depicted in Figure 6. Post formatting and de-duplication, we conduct multiple rounds of high-order mutations on both vulnerable and non-vulnerable code data.

In the first round, we determine which mutation operators to include or skip using random numbers, considering the presence of loops or branch structures in the code. We decide whether to mutate a specific point based on random probability, leading to exponential growth in mutated sample data after each round, as demonstrated in Figure 7.

Following several rounds of random mutation, we calculate the Self-BLEU diversity index for the mutation samples generated in each round. Upon approaching optimal diversity within the dataset, we select the first approaching position vector as the optimal mutation operator path. Leveraging this mutation operator order, we generate the final enhanced sample data for subsequent training of deep learning detection models.

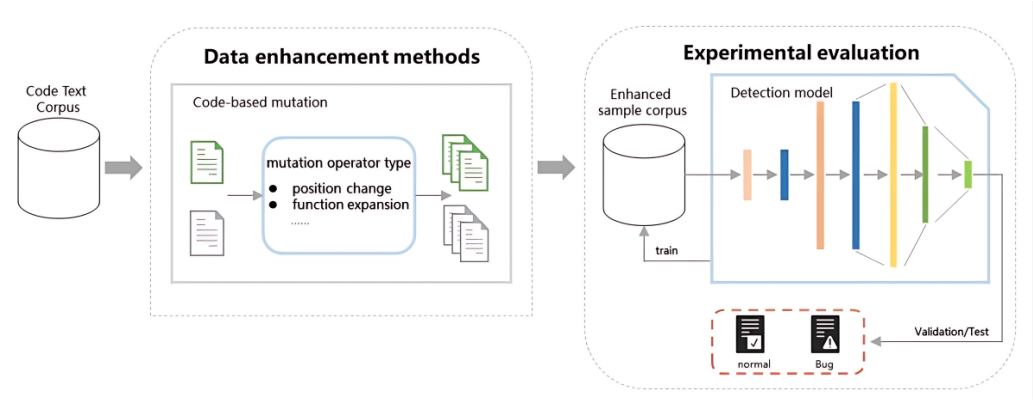

Figure 4. Overall framework of data enhancement

```
m static BlockDriverState *bdrv_append_temp_snapshot(BlockDriverState *bs,
m                                    int flags,
m                                    QDict *snapshot_options,
m                                    Error **errp)
m{
-   /* bdrv_append() consumes a strong reference to bs_snapshot (i.e. it will
-    * call bdrv_unref() on it), so in order to be able to return one, we have
-    * to increase bs_snapshot's refcount here */
    bdrv_ref(bs_snapshot);
    bdrv_append(bs_snapshot, bs, &local_err);
    if (local_err) {
        error_propagate(errp, local_err);
        ret = -EINVAL;
        goto out;
    }
-
    g_free(tmp_filename);
    return bs_snapshot;
-
out:
    QDECREF(snapshot_options);
    g_free(tmp_filename);
    return NULL;
}
```

a

```
static BlockDriverState *bdrv_append_temp_snapshot(BlockDriverState *bs, int flags, QDict *snapshot_options, Error **errp) {
    bdrv_ref(bs_snapshot);
    bdrv_append(bs_snapshot, bs, &local_err);
    if (local_err) {
        error_propagate(errp, local_err);
        ret = -EINVAL;
        goto out;
    }
    g_free(tmp_filename);
    return bs_snapshot;
out:
    QDECREF(snapshot_options);
    g_free(tmp_filename);
    return NULL;
}
```

b

Figure 5. Code formatting

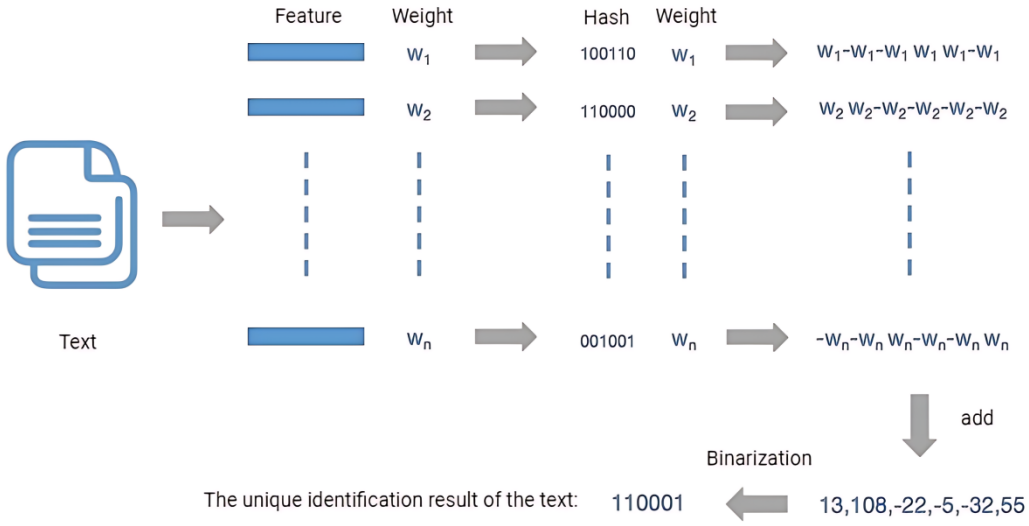

Figure 6. SimHash algorithm

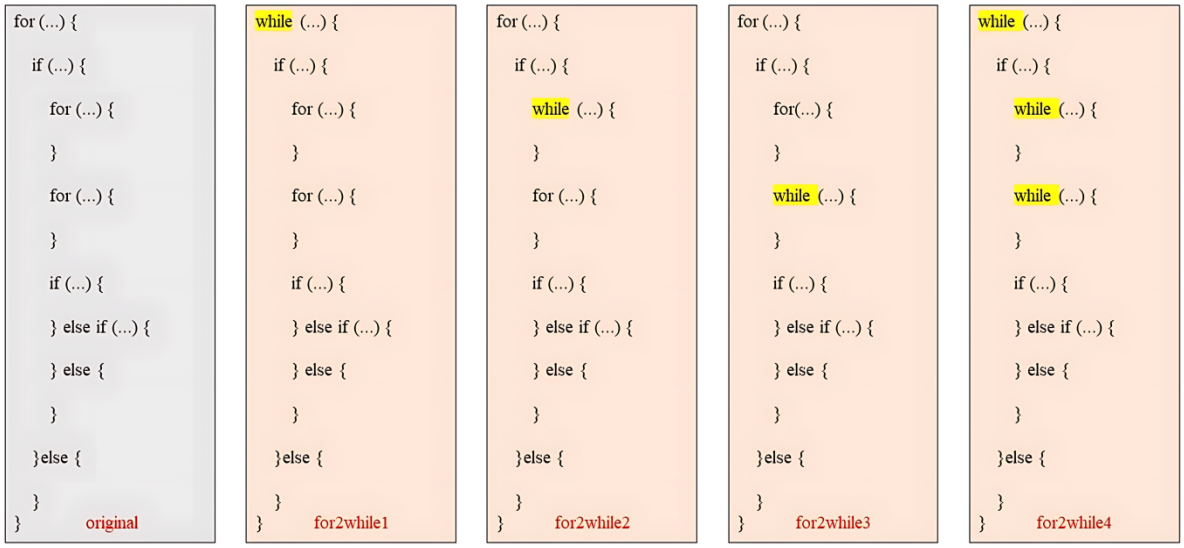

Figure 7. Mutation operator can generate multiple samples due to multiple mutation points in the code

3.3 Data Enhancement Based on Code Variation

Data enhancement techniques have matured in image and natural language processing, but there's a need for theory specific to code data enhancement. Code data, while fundamentally text-based, differs from natural language due to its structured syntax. Some methods from image and text processing can be adapted for code data enhancement. However, code data differs from images because it's non-serialized and context sensitive. Conventional image transformation methods don't apply to code fragments. Additionally, converting code into images for enhancement isn't feasible.

Code data shares similarities with text data but differs significantly in three key aspects: complexity, vocabulary, and structure. To enhance code data, real-world vulnerability datasets are crucial. These datasets should ideally contain diverse and consistent samples of the same vulnerability types. This paper's research direction focuses on enhancing vulnerability code data while maintaining semantic consistency to improve intelligent vulnerability detection accuracy.

4 Experimental Evaluation

4.1 RNN Models

We propose a data enhancement method with different processing granularities for vulnerability-related code. It first generates a variety of mutation operators that can maintain the semantic invariance of the vulnerable code. Then, it performs high-level mutation on the vulnerable code to expand the training samples. This section conducts experiments to verify the effectiveness of our data enhancement approach. We raise 5 research questions to guide our experiments, and each experiment focuses on different research questions.

4.2 Datasets

Two real-world open-source project vulnerability datasets are used in this paper's experimental research on

security vulnerability detection. Firstly, the data set used and published by Design is used to test the effect of the model. At the same time, to compare and study the two data enhancement methods proposed in this paper and to verify the detection effect of data enhancement on different data sets, auxiliary Validate with the REVEAL dataset.

Both Devign and REVEAL are derived from real-world open-source projects. They differ from synthetic and semi-synthetic datasets such as SARD regarding code complexity and readability. The Design dataset is obtained in several large open-source projects such as Linux, FFmpeg, QEMU, and Wireshark. Still, the paper's authors only publish the QEMU and FFmpeg datasets. Based on the above actual situation, and to compare with the experimental results of the predecessors, QEMU and FFmpeg are combined into one data set, which is consistent with the practice of the author of Devign's paper. The overview of the Design dataset is shown in Table 1. The number of submissions of vulnerabilities and non-vulnerabilities in this dataset is relatively ideal, and the ratio of positive and negative samples is relatively even. For the sample extraction method, the author of the paper extracted the critical functions as samples by analyzing the submitted information and assisted by manually marking them as vulnerability samples or non-vulnerability samples.

The REVEAL dataset selects the Linux Debian version project and Google's open-source Chromium project as data sources. REVEAL looks for security issues from the two fields of operating system engineering and browser engineering. In addition, REVEAL also uses a security keyword filtering method similar to Devign, which has improved the dataset's quality. As for the sample content, although REVEAL also extracted the code corresponding to the security vulnerability and its repair patch code and marked it at the function granularity, it also made some changes. An important point is functional analysis. For a vulnerability and its corresponding repair code, if it only involves the code content of one function, it can be directly compared and marked, but if the repair of a vulnerability consists of the range of multiple parts (especially if there are iterations), then the function that was initially marked as non-vulnerable needs to be re-marked. According to the original author's statistics, more than 80$\%$ of the vulnerability fixes involve multiple function modifications. As shown in Table 2, the REVEAL dataset has the problem of unbalanced sample distribution, but this paper needs to do it as an auxiliary verification dataset—excessive demands.

Table 1. Devign dataset overview

Project	Vulnerability related commits	Non-vulnerability related commits
FFmpeg	5962	8000
QEMU	4932	6978
Total	10894	14978

Table 2. REVEAL dataset overview

Dataset	Vulnerability related code	Non-vulnerability related code
REVEAL	2276	20558

Table 3. Experimental environment detailed parameter configuration table

Hardware configuration	Processor model	Intel(R) Xeon(R) Sliver 4110 CPU @2.10GHz 2.10GHz
	Memory size	64GB
	Disk size	4TB
OS	Windows 10(64-bit)	
Software configuration	Python 3.6.2, Pytorch 1.3.1, Anaconda 4.3.30	

Table 4. Training parameter configuration table

Parameter name	Parameter value	Remark
Max_Len	64	Slice length
Test_Size	0.2	Proportion of test set
Seed	20221116	Random seed
LR	1e-3	Learning rate
Weight_Decay	0	Weight decay
Emb_Dim	128	Vector length
Batch_Size	16	Single training sample number
Num_Epoches	10	Iterations
Input_Dim	64	Input feature dimension
Hidden_Size	256	Hidden state feature dimension
Dropout	0.2	Regularization parameter

Table 5. Devign dataset division

The number of samples	Training set	Validation set	Test set
Related to the vulnerability	10018	1187	1255
Related to the non-vulnerability	11836	1545	1477
Total	21854	2732	2732

Table 6. Data generated by a single mutation of the Devign training set

Variation method	Numbers of samples related to the vulnerability	Numbers of samples related to the non-vulnerability
$Devign_{origin}$	10018	11836
$Devign_{RDS}$	31732	29763
$Devign_{IF}$	23765	27112
$Devign_{CR}$	16753	16463

4.3 Environment and Evaluation Indicators

4.3.1 Programming Environment

The system uses Python programming, and Table 3 shows the configuration information of the server.

To facilitate the use of deep learning for data set experiments, the model training mainly uses the deep learning framework provided by Pytorch, Python programming, and C++ language for the development of index storage and retrieval

4.3.2 Software Vulnerability Detection Model

To evaluate the data enhancement method of this topic, it is necessary to implement a vulnerability classification detection model for verification results and then input the augmented corpus (including accurate data and sample data) into the classifier for training.

(1) Model Building

This topic constructs four neural networks, which are recurrent neural network (RNN), bidirectional recurrent neural network (BiRNN), long short-term memory network (LSTM) and bidirectional long short-term memory network (BiLSTM).RNNs were chosen for their ability to capture temporal dependencies, while BiRNNs excelled at contextual understanding in both forward and backward directions. LSTMs were employed to model intricate long-range dependencies, crucial for complex code structures. BiLSTMs combine bidirectional processing and long-term dependency modeling. Each selection was based on their individual strengths observed in natural language processing and sequence data analysis, ensuring a holistic vulnerability detection approach.

(2) Loss Function

The optimization algorithm we choose here is the Adam (Adaptive Moment Estimation) algorithm, which is essentially RMSprop with a momentum item d. It can dynamically adjust the learning rate of each parameter by using the first-order moment estimation and second-order moment estimation of the gradient.

(3) Train

This project divides the coded corpus into two parts, 80% of the data is used for training (including data enhancement samples), and 20% is used for testing. Then we use the training set for data enhancement. The trained classifier is then tested on samples from the test set, and its accuracy is recorded. There are mainly 11 hyperparameters in the training phase. After continuous experiments, we found the optimal parameter combination, as shown in Table 4.

4.3.3 Evaluation Indicator

In machine learning, the measurement and evaluation of models requires a reliable metric to facilitate later model iteration and optimization. This topic belongs to a typical binary classification problem. There are three leading measurement indicators: precision, recall and F1-Score, the definitions are as below.

TP (True Positive): The prediction is Positive, and the answer is True Positive.

FP (False Positive): The prediction is Positive, and the answer is Negative, so it is false Positive.

TN (True Negative): The prediction is Negative, and the answer is true Negative.

FN (False Negative): The prediction is Negative, and the answer is Positive, so it is false.

Precision indicates that the model prediction is predicted as a loophole. Still, the number of samples the vulnerability accounted for the proportion of all samples with openness, as shown in formula 1.

$$Precisin = \frac{TP}{TP + FP} \tag{1}$$

The recall rate indicates that the model prediction is predicted as a loophole. Still, the number of vulnerability samples accounts for the proportion of all pieces with loopholes, as shown in formula 2.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Table 7. Mutation matrix for code mutation after 10 rounds of random selection

Mutation round	RDS	IF	CR	Generate sample size	Self-BLEU indicator
0	0	0	0	21854	-
1	1	1	1	72435	0.893
2	1	0	0	118563	0.912
3	0	1	1	403114	0.878
4	1	0	1	1512431	0.736
5	1	0	0	4837370	0.747
6	0	1	0	11125951	0.782
7	1	0	1	34490452	0.713
8	1	1	1	158656074	0.692
9	1	0	0	491833831	0.701
10	0	0	1	821362497	0.722

Table 8. Finally select the code mutation variation matrix for the exp6 experiment

Mutation round	RDS	IF	CR	Generate sample size	Self-BLEU indicator
0	0	0	0	21854	-
1	1	0	1	39132	0.813
2	0	1	0	59821	0.774
3	0	0	1	77469	0.693
4	**0**	**0**	**1**	**92435**	**0.719**
5	1	1	0	212631	0.731
6	0	1	0	297640	0.706
7	1	0	0	565517	0.698
8	0	0	1	1074482	0.733
9	0	0	1	3878883	0.727
10	1	0	0	7085096	0.719

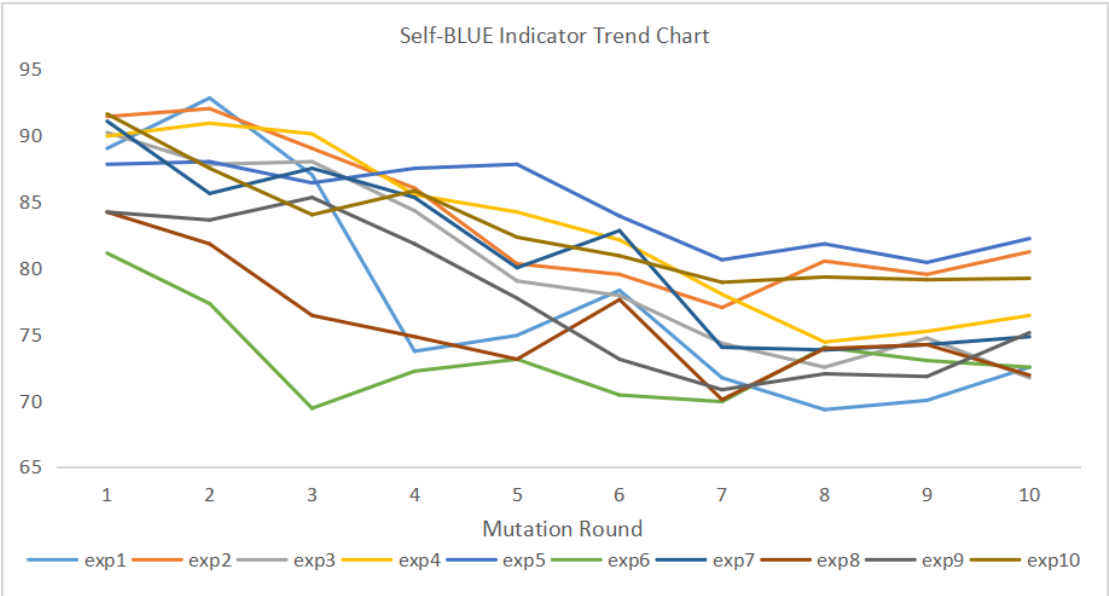

Figure 8. Convergence of data diversity index for high-order mutation of mutation operator selected randomly for ten times

F1-score represents the average F1 value of accuracy and recall rate, a comprehensive model indicator, as shown in formula 3.

$$\text{F1}-\text{score}=\frac{2\times Precision\times Recall}{Precision+Recall} \quad (3)$$

4.4 Research Questions

Our study aims to answer the following four research questions:

RQ1: Whether the data enhancement method based on mutation can effectively expand the training data set?

RQ1 is to verify whether the data enhancement method based on code mutation can generate many enhanced samples on the actual data set Design, which meets the primary conditions of data enhancement.

RQ2: Whether expanding the training data set by simple copying and duplication can improve the detection accuracy of the model?

RQ2 is a study to avoid the impact of a large amount of data on the model detection effect and generate training samples of the same magnitude by simple copying and replication for comparison.

RQ3: Whether the way of data enhancement can improve the detection accuracy of the model?

RQ3 is to study the enhanced samples generated by RQ1 and RQ2 and the unenhanced training samples together with multiple models for comparative experiments and compare the results to determine whether the detection accuracy of the model can be improved.

RQ4: Whether code mutation enhancement samples of different magnitudes can improve the detection accuracy of the model?

RQ4 is to study based on RQ1, RQ2, and RQ3 experiments, using random sampling, extracting enhanced samples of different sets of numbers and mixing them with the original data collection, and then inputting the model to study and analyze the detection accuracy of the model, and to judge the detection accuracy of different magnitudes Whether surrogate mutation augmentation samples can improve the detection accuracy of the model.

RQ5: Can the data enhancement method based on code mutation improve the detection accuracy of the model in other real data sets?

RQ5 studies the generalization verification of data augmentation in this paper. Generalization here refers to verifying whether the data augmentation method based on code mutation is effective on other real datasets and experimenting with the REVEAL dataset.

RQ6: whether the data enhancement method in this article can be used across projects?

RQ6 is to study whether the data enhancement method based on code variation can be used across projects and explore the possibility of its cross-project use by analyzing the forms and characteristics of this data enhancement.

5 Results and Analysis

5.1 Response to RQ1

RQ1: Whether the data enhancement method based on mutation can effectively expand the training data set?

In this section, we address RQ1 by exploring the impact of data enhancement methods on deep learning vulnerabilities using the cleaned DEVIGN dataset, as shown in Table 5. We applied two data enhancement methods to this dataset and evaluated their effect on vulnerability detection accuracy across four network models: RNN, BIRNN, LSTM, and Bilstm.

First, it is necessary to test whether the three mutation operators apply to Devign's original training set. The number of generated samples after a single code mutation is performed on the training set is shown in Table 6. The RDS mutation operator expanded the dataset by approximately 1.81 times, IF mutation by about 1.33 times, and CR mutation by roughly 0.52 times, demonstrating the feasibility of data enhancement through code mutation. Once we confirmed the effectiveness of these operators, we performed multiple high-order mutations guided by the Self-BLEU text diversity index to find the optimal mutation path. Table 8 displays the mutation matrix for three mutation operators subjected to a 10-order random selection for code mutation.

It's important to note that the generated samples in this stage serve as input for the next step, leading to a significant increase in the number of generated models. The Self-BLEU diversity index gradually converged to around 0.7 with the growing number of generated samples, as shown in Table 7. To determine the optimal mutation operator execution path and Self-BLEU convergence, we conducted ten rounds of random mutations. Table 8 presents the data variation matrices, and Figure 8 illustrates the convergence of the Self-BLEU index for these rounds.

Following ten high-level code mutations, we selected the most diverse mutation matrices to create the final mutation samples for the training set. Our experiments showed that generating a substantial amount of data through transformation had a limited impact on diversity. To balance the number of rounds and diversity, we chose the 'exp6' order from the mutation matrix in Table 8, resulting in the selection of 92,435 enhanced sample data after the fourth-order mutation.

In addressing RQ1, we significantly expanded the training data set while maintaining diversity. These diverse samples hold practical significance, enhancing the accuracy of vulnerability detection models across various network architectures. This methodological rigor not only contributes to academia but also promises valuable real-world applications in bolstering software security protocols.

Summary of RQ1

Through ten groups of 10-order code mutations of diversity indicators, a group of the most diverse mutation operators is selected for execution order and times. Ninety-

two thousand four hundred thirty-five enhanced sample data are generated using 21,854 original data, which meets the expansion requirements—basic requirements of the dataset.

5.2 Response to RQ2

RQ2: Whether expanding the training data set by simple copying and duplication can improve the detection accuracy of the model?

To address RQ2, this section processes 92,435 enhanced sample data from the divided training set of the Design dataset using code mutation. To mitigate the impact of significantly increasing the training set, a control experiment introduced enhanced samples that replicated and shuffled the initial training set. The training parameters are consistent with the software vulnerability detection model mentioned earlier. The three enhanced sample sets, $Devign_{origin}$, $Devign_{mutate}$, and $Devign_{copy}$, are input into RNN, BiRNN, LSTM, and BiLSTM models for training. Ten rounds of iterative testing were conducted on the test set in binary classification experiments, and results were statistically analyzed and visualized using box plots in Figure 9 to Figure 11.

Table 9 presents the index values of the three training sets after training. $Devign_{origin}$ serves as the baseline, while $Devign_{mutate}$ and $Devign_{copy}$ are the enhanced training sets. Results indicate that although $Devign_{copy}$ contains a similar number of samples as $Devign_{mutate}$, F1 scores across the three vulnerability detection models remain around 0.55. This suggests that simply copying and expanding the training set is insufficient to enhance the vulnerability detection model's effectiveness.

Examining RQ2, we amplified the training dataset by duplicating samples and analyzed their impact on model accuracy. Despite similar sample numbers, F1 scores around 0.55 indicated limited effectiveness. Simple copying fell short, emphasizing the necessity for more sophisticated methods in model improvement.

Summary of RQ2

The effect of expanding the training set samples by simple copy and replication on the model is negligible compared to using training samples without data enhancement.

5.3 Response to RQ3

RQ3: Whether the way of data enhancement can improve the detection accuracy of the model?

To answer RQ3, the data in this part is obtained from the previous experiments. Observation Table 9 shows the values of various indicators after the training of the three training sets. The F1 score of the detection model using $Devign_{mutate}$ is 14.79% higher than that of the model using $Devgin_{origin}$, which is accurate. The rate has increased by 21.73%, and the recall rate has not improved much compared with the other two indicators, which is 8%.

In addressing RQ3, our experiments reveal significant enhancements in the detection model's F1 score, the overall results underscore the effectiveness of our proposed code mutation-based data enhancement method in enhancing binary classification tasks.

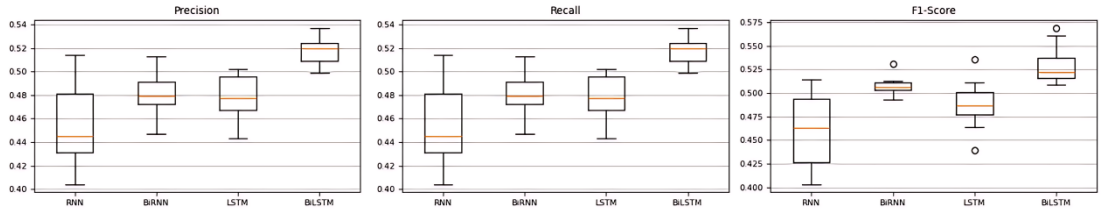

Figure 9. $Devign_{origin}$ dataset deep learning vulnerability detection experiment box plot

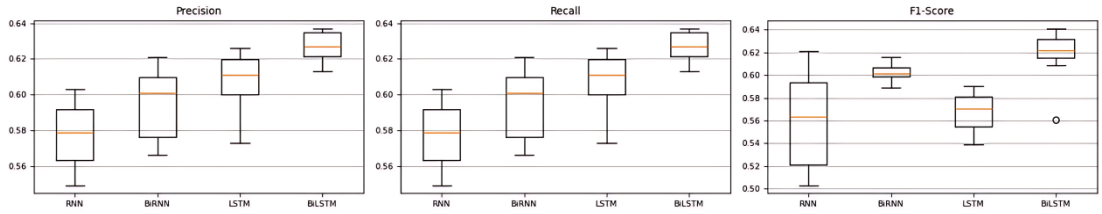

Figure 10. Box plot of deep learning vulnerability detection experiment using $Devign_{mutate}$ training set

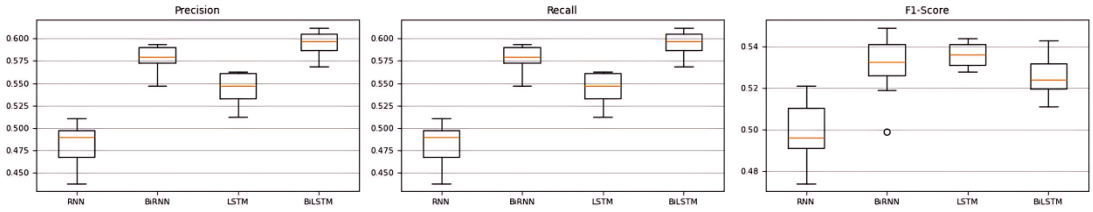

Figure 11. Using the $Devign_{copy}$ box training set for deep learning vulnerability detection experiment box line

Table 9. Using the Devgin dataset to study the effect of data enhancement on deep learning vulnerability detection research experiment results

Training dataset	Model	Precision	Recall	F1-Score
$Devign_{origin}$	RNN	0.491	0.541	0.514
	BiRNN	0.513	0.562	0.536
	LSTM	0.502	0.563	0.531
	BiLSTM	0.537	0.604	0.569
$Devign_{mutate}$	RNN	0.603	0.641	**0.621**
	BiRNN	0.621	0.563	**0.590**
	LSTM	0.626	0.607	**0.616**
	BiLSTM	0.637	0.644	**0.641**
$Devign_{copy}$	RNN	0.511	0.532	0.521
	BiRNN	0.594	0.502	0.544
	LSTM	0.563	0.537	0.549
	BiLSTM	0.612	0.489	0.543

Summary of RQ3

The data enhancement method proposed in this paper through code mutation can improve the experimental results of the binary classification task to a certain extent, reflected in multiple classification indicators.

5.4 Response to RQ4

RQ4: Whether code mutation enhancement samples of different magnitudes can improve the detection accuracy of the model?

To address RQ4, we examined the final part of the experiment. Our analysis revealed two key findings. Firstly, enhancing various aspects of code without changing semantics in the training set improves the accuracy of the detection model. Secondly, data enhancement contributes by increasing the training set's size, often leading to improved model accuracy due to the availability of more training data.

To investigate the impact of data augmentation on models with different training set sizes, we created enhanced samples of various sizes using Devginmutate from the previous section. The original data was randomly sampled to create training sets of different magnitudes (as shown in Table 10), ranging from 1N to 5N, underwent code mutation and ANN retrieval recall-based enhancement before being input into multiple network models to assess vulnerability detection accuracy.

Figure 12 illustrates how different training set sizes affect the F1 score of the detection model using the code mutation-based enhancement method. The x-axis represents the data size, ranging from 1N to 5N, achieved by mixing original samples with enhanced selections. The y-axis represents the F1 score, an evaluation index for the model. The figure shows a significant improvement in model performance for Devginmutate enhanced models with smaller samples. However, this performance improvement gradually diminishes as the size of enhanced samples increases. The optimal number of enhanced samples for Devginmutate models is approximately 3N, with further expansion not yielding better results.

Table 10. Different magnitudes of training set data distribution

Training set size	$Devgin_{mutate}$ Sampling number
1N	18487
2N	36974
3N	55461
4N	73948
5N	92435

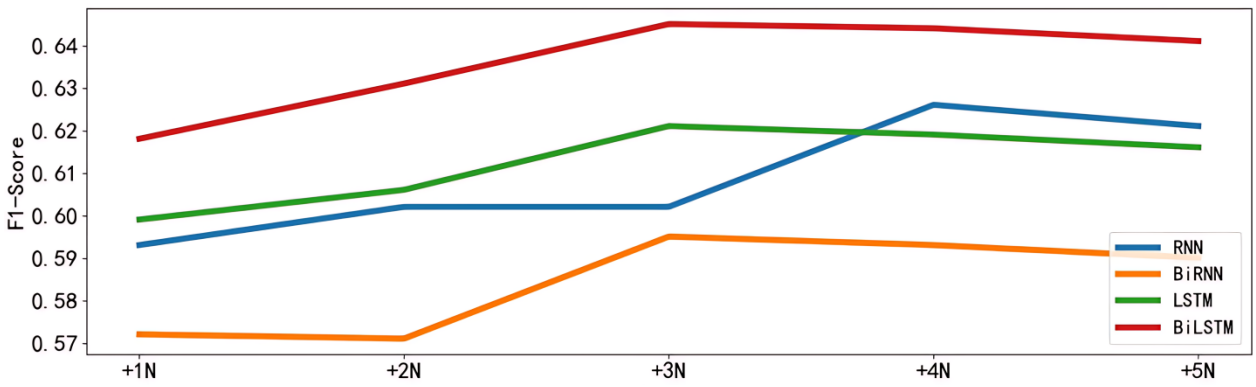

Figure 12. The influence of different magnitudes of enhanced samples on the model effect under the method of code mutation enhancement

Our investigation into varying data augmentation magnitudes (1N to 5N) yielded crucial insights. Enhancing different aspects of code, without altering semantics, significantly improved detection accuracy. Furthermore, augmenting data increased training set size, enhancing model accuracy due to richer training data. These findings emphasize the delicate balance required for effective data augmentation, particularly in resource-constrained scenarios.

Summary of RQ4

Although the variation range of the model's evaluation index F1 score is small (less than ten percentage points), it can be found that in the case of small samples and low resources, the data enhancement method in this paper can effectively improve the accuracy of the model to a certain extent, but adding too many augmented samples may not help much.

5.5 Response to RQ5

RQ5: Can the data enhancement method based on code mutation improve the detection accuracy of the model in other real data sets?

To answer RQ5, the experiments in this part use the REVEAL security vulnerability data set of the whole open-source project in the experimental research of security vulnerability detection. Because the data set has specific data class imbalance problems, the code samples containing vulnerabilities and the code samples without Vulnerable code samples should be as balanced as possible in number. Table 11 shows the division of the balanced training set, verification set, and test set, and then this experiment will conduct data enhancement experiments on the divided training set.

Do the same data enhancement processing for the RQ1 part of the REVEAL training set after sampling and division, and the number of $REVEAL_{mutate}$ in the enhanced sample set is shown in Table 12.

Input the enhanced sample sets $REVEAL_{origin}$ and $REVEAL_{mutate}$ described above in this section into the RNN, BiRNN, LSTM, and BiLSTM models for training. Since the training process is similar to the experiment in RQ1, Table 13 directly shows the experimental results.

Table 13 shows the results of model-checking experiments on the REVEAL training set after enhancement. Because the scale of the REVEAL dataset itself is not large, even if the samples are enhanced and expanded, the improvement of the detection effect of the model is limited. However, compared with $REVEAL_{origin}$, the accuracy of $REVEAL_{mutate}$ increased by about 16.86%, the recall rate increased by approximately 24.86%, and the F1 score increased by approximately 21.78%.

In addressing RQ5, we extended our study to the REVEAL security vulnerability dataset. Despite the dataset's inherent limitations due to class imbalances, our code mutation-based data augmentation proved effective. These consistent enhancements underscore the method's stability across diverse datasets, showcasing its generalizability in improving model performance. This broad applicability enhances the method's practical utility in real-world vulnerability detection scenarios.

Table 11. REVEAL data set sampling division system situation

	Training set	Verification set	Test set
Numbers of samples related to the vulnerability	1822	227	227
Numbers of samples related to the non-vulnerability	1778	233	233
Total	3600	460	460

Table 12. REVEAL training set for data enhancement

	Vulnerability related samples	Non-vulnerability related samples	Total
$REVEAL_{origin}$	1822	1778	3600
$REVEAL_{mutate}$	7664	7467	15131

Table 13. Experimental results of research on the effect of data enhancement on deep learning vulnerability detection

Training dataset	Model	Precision	Recall	F1-Score
$REVEAL_{origin}$	RNN	0.277	0.339	0.305
	BiRNN	0.346	0.267	0.301
	LSTM	0.487	0.403	0.441
	BiLSTM	0.491	0.371	0.422
$REVEAL_{mutate}$	RNN	0.363	0.298	0.327
	BiRNN	0.489	0.414	0.448
	LSTM	0.497	0.518	0.507
	BiLSTM	0.522	0.493	0.507

Table 14. Analysis of data augmentation methods

	Code-based mutation
Granularity of operation	Codes
Operation logic	Vulnerable code is still vulnerable code after mutation
Whether across projects and languages	No

Summary of RQ5

The data augmentation method based on code mutation has stable performance on multiple datasets, which shows that the data augmentation method proposed in this paper has certain generality.

5.5 Response to RQ6

RQ6: whether the data enhancement method in this article can be used across projects?

To answer RQ6, the data in this part is based on the evaluation of the effectiveness of the RQ1 data enhancement method and the analysis of the enhancement experiment on the RQ5 auxiliary data set REVEAL. The data enhancement method based on code mutation is effective on multiple data sets and detection models. Solid performance. Table 14 is used to analyze the data enhancement method based on code mutation to explore its applicable specific scenarios.

From the analysis of the above Table 14, it can be seen that the data enhancement method of the code mutation sampled in this paper is mutated based on C language grammar and logic. Although the mutation strategy can be reproduced in other languages, the realization of cross-project mutation is currently Unrealistic.

Summary of RQ6

The data augmentation method based on code mutation to achieve cross-project transformation is currently not realistic.

6 Conclusion and Future Work

This paper presents a mutation-based data enhancement approach to improve software vulnerability detection by generating diverse training samples through multiple mutation rounds. Our method significantly enhances the detection accuracy of deep neural networks when applied to open-source datasets, showcasing the effectiveness of code mutation in addressing data imbalance issues.

In the future, we plan to optimize mutation operators further and explore cross-project data enhancement. Additionally, expanding the application of this method to other programming languages and more complex software systems will be an important direction for future research.

References

[1] X. Wu, W. Zheng, X. Chen, Y. Zhao, T. Yu, D. Mu, Improving high-impact bug report prediction with combination of interactive machine learning and active learning, *Information and Software Technology*, Vol. 133, Article No. 106530, May, 2021.

[2] X. Wu, W. Zheng, X. Xia, D. Lo, Data quality matters: A case study on data label correctness for security bug report prediction, *IEEE Transactions on Software Engineering*, Vol. 48, No. 7, pp. 2541–2556, July, 2022.

[3] L. Bao, X. Xia, A. E. Hassan, X. Yang, V-SZZ: automatic identification of version ranges affected by CVE vulnerabilities, *2022 44th International Conference on Software Engineering (ICSE)*, Pittsburgh, PA, USA, 2022, pp. 2352–2364.

[4] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, M. Stadler, J. Weller, J. Kuhn, G. Kasneci, ChatGPT for good? On opportunities and challenges of large language models for education, *Learning and Individual Differences*, Vol. 103, Article No. 102274, April, 2023.

[5] S. Kim, S. Yoo, Multimodal surprise adequacy analysis of inputs for natural language processing DNN models, *2021 2nd IEEE/ACM International Conference on Automation of Software Test (AST)*, Madrid, Spain, 2021, pp. 80–89.

[6] T. Suzuki, Teachaugment: Data augmentation optimization using teacher knowledge, *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, LA, USA, 2022, pp. 10894–10904.

[7] E. Real, C. Liang, D. R. So, Q. V. Le, Automl-zero: Evolving machine learning algorithms from scratch, *2020 37th International Conference on Machine Learning (ICML)*, Virtual Event, 2020, pp. 8007–8019.

[8] C. Uhle, M. Torcoli, J. Paulus, Controlling the perceived sound quality for dialogue enhancement with deep learning, *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, 2020, pp. 51–55.

[9] P. Xu, Y. Wang, X. Chen, Z. Tian, Deep kernel learning networks with multiple learning paths, *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Singapore, 2022, pp. 4438–4442.

[10] L. Gazzola, D. Micucci, L. Mariani, Automatic software repair: A survey, *2018 40th International Conference on Software Engineering*, Gothenburg, Sweden, 2018, pp. 1219–1219.

[11] X. Kong, L. Zhang, W. E. Wong, B. Li, Experience Report: How Do Techniques, Programs, and Tests Impact Automated Program Repair? *2015 IEEE 26th International Symposium on Software Reliability Engineering*, Gaithersburg, Maryland, USA, 2015, pp.

194–204
[12] G. Spanos, L. Angelis, A multi-target approach to estimate software vulnerability characteristics and severity scores, *Journal of Systems and Software*, Vol. 146, pp. 152–166, December, 2018.
[13] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, Sysevr: A Framework for Using Deep Learning to Detect Software Vulnerabilities, *IEEE Transactions on Dependable and Secure Computing*, Vol. 19, No. 4, pp. 2244–2258, July-August, 2022.
[14] J. Coffman, A. Chakravarty, J. A. Russo, A. S. Gearhart, Quantifying the Effectiveness of Software Diversity Using Near-Duplicate Detection Algorithms, *2018 5th ACM Workshop on Moving Target Defense*, Toronto, ON, 2018, pp. 1–10.
[15] D. Fraunholz, D. Krohmer, S. D. Antón, H. D. Schotten, Catch Me If You Can: Dynamic Concealment of Network Entities, *2018 5th ACM Workshop on Moving Target Defense*, Toronto, ON, 2018, pp. 31–39.
[16] D. Weiss, C. Alberti, M. Collins, S. Petrov, Structured Training for Neural Network Transition-based Parsing, *53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, Beijing, China, 2015, pp. 323–333.
[17] A. P. B. Veyseh, T. H. Nguyen, D. Dou, Graph based Neural Networks for Event Factuality Prediction using Syntactic and Semantic Structures, *57th Conference of the Association for Computational Linguistics*, Florence, Italy, 2019, pp. 4393–4399.
[18] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, L. Wang, Every Document Owns Its Structure: Inductive Text Classification via Graph Neural Networks, *2020 58th Annual Meeting of the Association for Computational Linguistics*, Online, 2020, pp. 334–339.
[19] B. Bogin, J. Berant, M. Gardner, Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing, *2019 57th Conference of the Association for Computational Linguistics (ACL)*, Florence, Italy, 2019, pp. 4560–4565.
[20] W. E. Wong, Mutation Testing for the New Century, *Springer Science & Business Media*, 2001.
[21] S. Chakraborty, R. Krishna, Y. Ding, B. Ray, Deep learning based vulnerability detection: Are we there yet?, *IEEE Transactions on Software Engineering*, Vol. 48, No. 9, pp. 3280–3296, September, 2022.

Biographies

Lianyi Su (2002-), female, master, School of Software, Northwestern Polytechnical University. The main research area is formal validation.

Jie Hu (1998-), male, master, School of Physics and Technology, Yangzhou University. The main research areas are video processing and data enhancement algorithms.

Wei Zheng (1975-), male, doctor, Associate professor, Senior member of CCF, The main research areas are software testing, software security, and software warehouse excavate.