Mutation Operator Reduction for Cost-effective Deep Learning Software Testing via Decision Boundary Change Measurement

Li-Chao Feng[1], Xing-Ya Wang[1, 2*], Shi-Yu Zhang[1], Rui-Zhi Gao[3], Zhi-Hong Zhao[1]

[1] College of Computer Science and Technology, Nanjing Tech University, China
[2] Command and Control Engineering College, Army Engineering University of PLA, China
[3] Sonos Inc., USA
lcfeng@njtech.edu.cn, xingyawang@outlook.com, syzhang0825@njtech.edu.cn,
ruizhi.gao@sonos.com, zhaozhih@njtech.edu.cn

Abstract

Mutation testing has been deemed an effective way to ensure Deep Learning (DL) software quality. Due to the requirements of generating and executing mass mutants, mutation testing suffers low-efficiency problems. In regard to traditional software, mutation operators that are hard to cause program logic changes can be reduced. Thus, the number of the mutants, as well as their executions, can be effectively decreased. However, DL software relies on model logic to make a decision. Decision boundaries characterize its logic. In this paper, we propose a DL software mutation operator reduction technique. Specifically, for each group of DL operators, we propose and use *DocEntropy* to measure the model's decision boundary changes among mutants generated and the original model. Then, we select the operator group with the highest entropy value and use the involved operators for further mutation testing. An empirical study on two DL models verified that the proposed approach could lead to cost-effective DL software mutation testing (i.e., 33.61% mutants and their executions decreased on average) and archive more accuracy mutation scores (i.e., 9.45% accuracy increased on average).

Keywords: DL software, Mutation testing, Decision boundary, Mutation operator reduction

1 Introduction

Recently, Deep Learning (DL) software has been widely used in various safety-critical areas (e.g., face unlocking [1] and autonomous driving [2]). The defects in DL software may lead to disastrous consequences such as privacy leaks or car accidents. Therefore, DL software should be thoroughly tested [3]. Mutation testing is a conventional defect introducing based test adequacy measurement method [4], and it has been deemed an effective means to evaluate the adequacy of DL software testing [5]. We obtain a series of mutants by using mutation operators, and then we detect the defects in these mutants. In regard to DL software, researchers have proposed eight source-level mutation operators and eight model-level mutation operators [5]. The former works on the training set or the program, while the latter works on the trained model. Each operator can generate plenty of mutants. For each mutant, testers should record its executing results on all test data. However, it costs too much time to complete a DL mutation testing. Take the hand-written electronic dataset MNSIT [6] as an example. It contains 10,000 test data. Assume applying one DL mutation operator can generate ten mutants; we'll get 160 mutants. To complete the MNSIT mutation testing, we must conduct at least 1.6 million tests. These facts show that DL mutation testing suffers low-efficiency problems, making it difficult to use in practice.

Generally speaking, the evaluation indicators of mutation testing are the cost of test overhead (the number of mutants generated and executed) and the mutation score [7]. Mutation operator reduction is an effective means to reduce the size of mutants and improve the efficiency of mutation testing [8]. In regard to traditional software, its executing result is determined by program logic [9]. To conduct an adequate test, the logic differences among the source and mutated programs should be diverse. Then, testers can determine whether the test data can detect various defects at different locations. For example, by modifying the relational operators in each branch statement, testers can fully evaluate the ability of test data to detect boundary defects [10]. To speed up the mutation testing of traditional software, mutation operators that are difficult to cause program logic change can be reduced. Unlike traditional software, the trained model determines the executing result of DL software [11]. The training program fits the data features to obtain the decision boundaries of the model. All decision boundaries constitute the logic of the model [12]. Take a two-classification model as an example. The decision boundary divides two data classes into their respective decision spaces, which completes the data classification. Therefore, to evaluate the defect detection capabilities of DL test data, the difference of the decision boundary among the source program and mutants should be as diverse as possible.

In this paper, we propose a DL mutation operator reduction method which relies on decision boundary change measurement to select efficient mutation operators. In regard to DL software, we first quantify the difference of the decision boundary between the source program and each of the mutants based on Manhattan distance [13].

Subsequently, we propose Decision Boundary Change Entropy (*DocEntropy*) and use it to measure the decision boundary change diversity of a set of generated mutants. Finally, we select the operator group with the highest *DocEntropy* values as the reduced result and use it for further

*Corresponding Author: Xing-Ya Wang; E-mail: xingyawang@outlook.com
DOI: 10.53106/160792642022052303018

mutation testing. The main contributions of this paper are as follows:

- We propose the first DL software mutation operator reduction method. Specifically, we introduce decision boundary change and propose *DocEntropy* to measure the diversity of changes to select efficient mutation operators.
- We verify the effectiveness of the proposed method through empirical study. The experiment results on two DL models show that the technique can reduce 13 mutation operators to 8, decreasing the average of 33.61% mutants generated and executed. Moreover, it improves nearly 9.45 % mutation score accuracy on two models.

The remaining structure of this paper is organized as follows. Chapter 2 introduces the background of the DL model and its decision boundary, as well as DL mutation. Chapter 3 details the proposed reduction method. In Chapter 4, an empirical study is carried out. Chapter 5 introduces related work. Finally, the last Chapter summarizes our work.

2 Background

2.1 DL Model and Its Decision Boundary

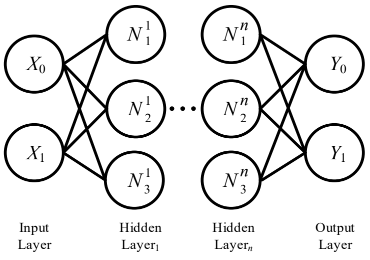

Figure 1. General structure of DL model

As mentioned earlier, DL software relies on a trained model to make decisions. A DL model is a three-level Neuron-Layer-Model structure [14]. As shown in Figure 1, it contains one input layer, one output layer, and several hidden layers. Each layer has a series of neurons. The neurons in the adjacent layers are connected.

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{1}$$

Neurons are the primary computing units of the DL model, each of which includes one linear transformation and one activation function. Its output, y, is a continuous variable. Formula (1) describes the structure of a neuron. w_i and b are the weights between neurons, which are the trainable parameters in the model. Their values are obtained from the training process. The activation function, $f()$, is the key to realize feature extraction because it can capture the nonlinear changes in the model.

To get a trained model, we first need to collect a training data set and write a training program. The former provides the learned characteristic, while the latter contains the structure of the model and artificially defined hyper-parameters. Then, we input the data set into the program, which fits the data parameters, and finally get the trained model.

The decision boundaries constitute the internal logic of the machine learning model. Regarding the DL model, the output layer describes its decision boundaries. Assume the outputs of all neurons in the output layer are $y = \{y_1, y_2, \ldots, y_k\}$, y_i denotes the probability of classifying the data into class *i* [15]. It satisfies $\sum_{i=1}^{k} y_i = 1$. If y_i archives the highest value, the data would be classified into class *i* by the decision boundaries.

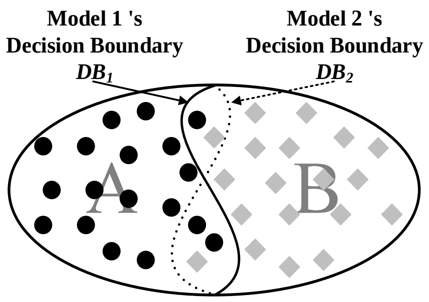

Figure 2. An example of decision boundaries in the DL model

For example, regarding the two-classification problem, data distributing in the decision space is divided into two classes by the DL decision boundary. Figure 2 illustrates two DL decision boundaries, DB_1 and DB_2. Each of them divides the data into two classes (i.e., C_1^A and C_1^B, C_2^A and C_2^B). One data in C_1^A and one in C_1^B are misclassified by DB_1, while two in C_2^A and one in C_2^B are misclassified by DB_2. It indicates that the decision boundary directly influences the classified results, and data close to the decision boundary are intuitively easier to misclassify.

2.2 DL Mutation Testing

Mutation testing was firstly proposed to assess the quality of the DL test set in 2018 [5]. It generates a large number of mutants through mutation operators. If the test data can kill more mutants, the dataset's quality is higher. The workflow of DL mutation testing is shown in Figure 3. It includes mutating, training, and testing. The former chooses operators and generates mutated models (i.e., mutants). The latter calculates the mutation scores by original model and mutants.

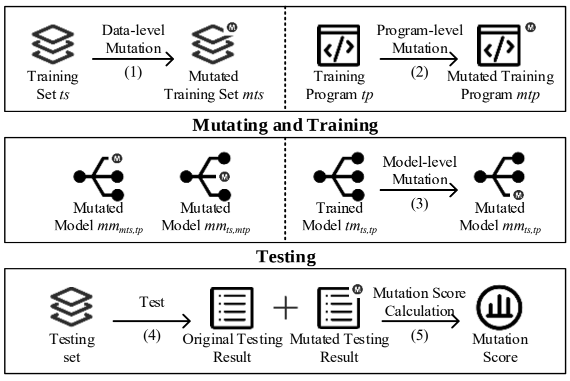

Figure 3. Workflow of DL mutation testing

Table 1. DL mutation operators

Mutation Stage	Mutation Object	Mutation Operator	Mutation Operator Description
Source-level (Before training)	Data-level (MO_d)	Data Repetition (DR)	Duplicate training data
		Label Error (LE)	Falsify results (e.g., labels) of data
		Data Missing (DM)	Remove selected data
		Data Shuffle (DF)	Shuffle selected training data
		Noise Perturb. (NP)	Add noise to training data
	Program-level (MO_p)	Layer Removal (LR)	Remove a layer before training
		Layer Addition (LAs)	Add a layer before training
		Activation Function Removal (AFRs)	Remove activation functions before training
Model-level (After training)	Neuron-level (MO_n)	Gaussian Fuzzing (GF)	Fuzz weight by Gaussian Distribution
		Weight Shuffling (WS)	Shuffle selected weights
		Neuron Switch (NS)	Switch two neurons of the same layer
		Neuron Effect Block (NEB)	Block a neuron effect on following layers
		Neuron Activation Inverse (NAI)	Invert the activation status of a neuron
	Layer-level (MO_l)	Layer Deactivation (LD)	Deactivate the effects of a layer
		Layer Addition (LAm)	Add a layer in the neuron network after training
		Activation Function Removal (AFRm)	Remove activation functions after training

Steps (1), (2), and (3) in Figure 3 are the process of mutating and training. Testers apply data-level mutation operators to the training set *ts*, generating the mutated training set *mts.* Testers apply program-level mutation operators to the source program *tp*, generating the mutated training set *mtp.* Then, the mutated model is trained by *mts* and *tp*, or *ts* and *mtp*. Besides, testers also can apply model-level mutation operators to the origin model, trained by *ts and tp* in advance. After generating the mutant set, the data is input into original and mutation models to obtain corresponding results by steps (d) and (e). Finally, testers calculate the mutation score, which reflects the quality of test data.

Mutation operator plays the role of generating kinds of mutants. Table 1 summarizes the characteristics of DL mutation operators [5]. For each operator, its mutation stage (column 1), mutation object (column 2), name (column 3), and a brief description (column 4) are described. According to the scopes, including the stage and object, mutation operators can be divided into data-level, program-level, neuron-level, and layer-level ones. The specific classification information is as follows:

(1) Mutation operators at data-level (MO_d). They act on the data of the training set, including five types: Data Repetition (DR), Label Error (LE), Data Missing (DM), Data Shuffle (DF), and Noise Perturb (NP). They change the characteristics of single or multiple data of the set, the distribution of the data set.
(2) Mutation operators at program-level (MO_p). They act on the training program, including three types: Layer Removal (LR), Layer Addition (LA_s), and Activation Function Removal (AFR_s). They change the structure settings of the model in the training program.
(3) Mutation operators at neuron-level (MO_n). They act on the neurons in the trained model, including five types: Gaussian Fuzzing (GF), Weight Shuffling (WS), Neuron Switch (NS), Neuron Effect Block (NEB), and Neuron Activation Inverse (NAI). They change the corresponding parameters of neurons.
(4) Mutation operators at layer-level (MO_l). They act on the layers in the trained model, including three types: Layer Deactivation (LD), Layer Addition (LA_m), and Activation Function Removal (AFR_m). They change the relevant information of the middle layer of the model.

3 Reduction Method

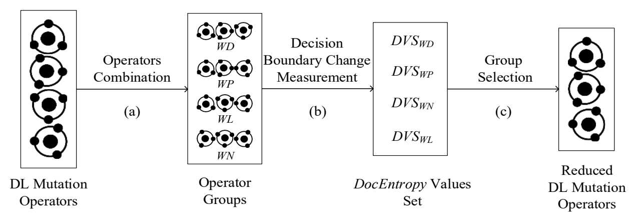

Figure 4. Framework of the proposed DL mutation operation reduction method

Figure 4 presents the framework of the proposed DL mutation operator reduction method. As previously mentioned, its basis corresponds to maximizing the diversity among the decision boundary changes of the original DL model and its mutated ones. Thus, it requires a strategy of operator combination to generate a series of candidate operator groups and a measurement of change diversity for sets of DL models. Therefore, our method works with the following two main phases: (a) Operator combination. We ignore one level DL mutation operator each time, and we get four operator groups. (b) Decision Boundary Change Measurement. We propose *DocEntropy* (i.e., Decision Boundary Change Entropy) to measure the diversity of boundary changes w.r.t. an operator group. Finally, we compare the *DocEntropy* values among all operator groups and select the operators included in the group that achieves the highest *DocEntropy* value on average as the DL mutation operator reduced result.

3.1 Operator Combination

Step F(a) in Figure 4 shows the specific operator subsets generation method. We ignore one level of mutation operators each time and combine the remaining levels of operators to generate the operator groups, i.e., the group without data-level operators ($MOG_{p,n,l}$), the group without program-level operators ($MOG_{d,n,l}$), the group without neuron-level operators ($MOG_{d,p,l}$), and the group without layer-level operators ($MOG_{d,p,n}$).

There are two factors that explain this operation. First, there is a specific contingency in single-type mutation operators, creating the low indicator stability. For example, there may be one good and one bad result of two mutants generated by one operator. Keeping as many levels of mutation operators as possible can make the quality of mutants more stable. Second, although only one mutation operator is used in one mutant, there is some potential relevance between the scopes of the mutation operators. For example, the scopes of MO_d and MO_p are the two interdependent objects of the training process. The mutation scores calculated by different levels of mutant combination are more meaningful.

In this paper, we refer to the reduction strategy for traditional mutation operators [16]. We generate operator groups by ignoring one of four levels operators MO_d, MO_p, MO_n, and MO_l. Compared with the single-level mutation operator group, the mutation operator group formed by the three levels has a larger cardinality. It can effectively alleviate the contingency caused by the number of mutation operators, making the result of the operator group more stable. Second, the remaining three levels of mutation operators can maintain the potential connection, making the method more comprehensive in the consideration of the nature of mutation operators.

3.2 Decision Boundary Change Measurement

As shown in Figure 4 (b), the corresponding changes in the decision boundary needs to be measured. Figure 5 illustrates the framework of decision boundary change measurement. It includes mutant generation, sample selection, and *DocEntropy* calculation.

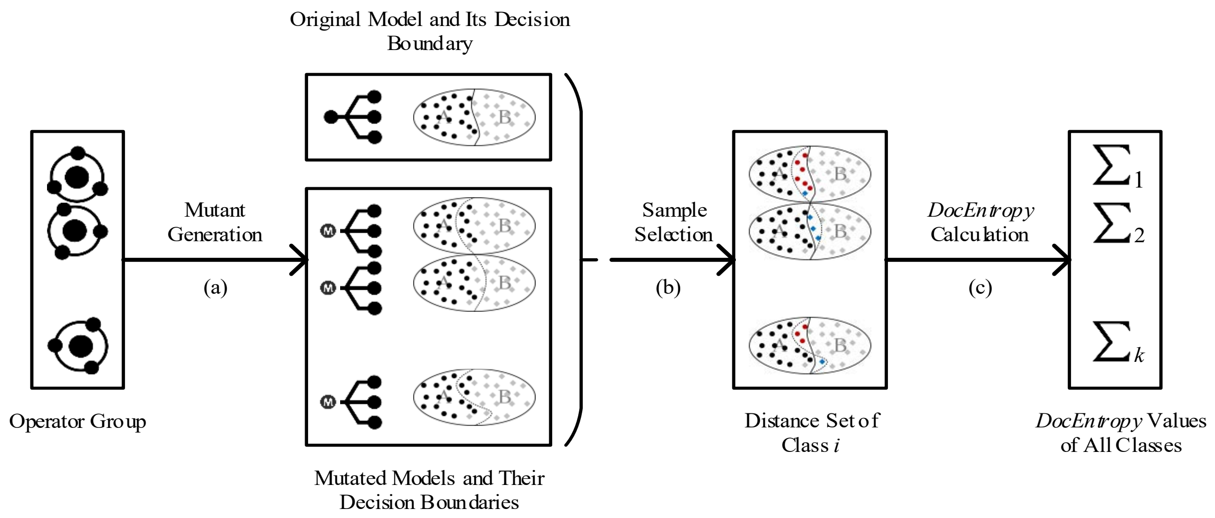
Original Model and Its Decision Boundary
Mutant Generation
(a)
Sample Selection
(b)
DocEntropy Calculation
(c)
Operator Group
Mutated Models and Their Decision Boundaries
Distance Set of Class *i*
DocEntropy Values of All Classes
A
B
$\sum_1$
$\sum_2$
$\sum_k$

Figure 5. Framework of decision boundary change measurement

Algorithm 1 outlines the details of three steps. For a task of *k* classification, it treats the number of classes *k*, a training set *TR*, the training program *TP*, the original model *m*, a testing set *TE*, and a mutation operator group *MOG* as inputs. It finally outputs a list of *DocEntropy* values *DVS*, where DVS_i denotes the diversity of decision boundary changes w.r.t. the i_{th} class.

Step 1, mutant generation (lines 1-12). Each operator in *MOG* generates the corresponding mutants, which are added to the mutant set *MUT*. In this process, objects that correspond to the operators in *MOG* are selected for mutating (lines 3-11).

Step 2, sample selection (lines 13-20). For the k classification task, test data *t* in class $i\,(1 \leq i \leq k)$ is iteratively selected for classification (line 16). If *t* is correctly classified by *m* but not correctly classified by m', the decision boundary of class *i* on m' changes (lines 17-18). Then, using formula (4), we calculate the boundary distance between *m* and m' (line 19) and $dis(t, i, m, m')$ is added to the distance set of the i_{th} class DS_i (line 20). After the distance calculations of all classes, DS represents the distances of all classes' decision boundaries on mutant set *MUT*. DS_i contains the distance of the decision boundary changes of class *i*.

The output of *t* at the i_{th} neuron on the output layer of *m* is y_i . For a k-classification problem, we assume the probability of classifying a data to each class is $1/k$. As shown in formulas (2) and (3), we use the Manhattan distance [13] to quantify the distance (i.e., d_1) between the decision boundary of *m* and data *t* on class *i*, as well as the distance (i.e., d_2) between the decision boundary of m' and data *t* on class *i*. The decision boundary distance of class *i* between *m* and m'on *t* is defined as formula (4).

Step 3, *DocEntropy* calculation (lines 21-24). For the distance set $DS_i\,(1 \leq i \leq k)$, we use *DocEntropy* to measure the degree of change of the decision boundary. $DocEntropy(DS_i, MUT)$is added to *DVS* (line 24).

Algorithm 1. Decision boundary change measurement

```
Input: $k$, $TR$, $TP$, $m$, $TE$, $MOG$
Output: $DVS$
1:  // Step 1: mutant generation
2:  initialize the mutant set $MUT = \{\}$;
3:  foreach operator $op$ in $MOG$:
4:      if $op \in MO_d$:    // data-level
5:          Mutate $TR$ to generate mutants $MUT_1$;
6:      else if $op \in MO_p$:    // program-level
7:          Mutate $TP$ to generate mutants $MUT_2$;
8:      else if $op \in MO_n$:    // neuron-level
9:          Mutate $m$ to generate mutants $MUT_3$;
10:     else:    // layer-level
11:         Mutate $m$ to generate mutants $MUT_4$;
12:     $MUT = MUT \cup MUT_1 \cup MUT_2 \cup MUT_3 \cup MUT_4$;
13: // Step 2: sample selection
14: for $i$ in all classes (1, $k$):
15:     initialize the distance set of the $i_{th}$ class $DS_i = \{\}$;
16:     foreach test data $t$ in $TE$:
17:         foreach mutant $m'$ in $MUT$:
18:             if $m(t) = i$ and $m'(t) != i$:
19:                 calculate $dis(t, i, m, m')$;
20:                 add $dis(t, i, m, m')$ to $DS_i$;
21: // Step 3: DocEntropy calculation
22: for $i$ in all classes (1, $k$):
23:     calculate $DocEntropy(DS_i, MUT)$;
24:     add $DocEntropy(DS_i, MUT)$ to $DVS$;
25: output $DVS$.
```

$$d_1 = \left|y_i - \frac{1}{k}\right| \tag{2}$$

$$d_2 = \left|y'_i - \frac{1}{k}\right| \tag{3}$$

$$dis(t, i, m, m') = d_1 + d_2 \tag{4}$$

$$DocEntropy(DS_i, MUT) = \frac{dis_{sum}}{|MUT|} * \left(-\sum_{j=1}^{|DS_i|} \frac{dis}{dis_{sum}} \ln \frac{dis}{dis_{sum}}\right) \tag{5}$$

Formula (5) presents the calculation of *DocEntropy*. dis_{sum} denotes the sum of all distances in DS_i, $|MUT|$ denotes total number of mutants, and $|DS_i|$ denotes the number of distances in the set DS_i. Entropy is an indicator to measure the degree of disorder [17], which has been proved to be a better indicator to measure the variety of data than other measurement indicators [18]. The more chaotic the object, the greater the entropy value. The diversity of decision boundary changes helps find test data close to the boundary. In mutation testing, mutants whose decision boundaries close to the original ones could misclassify fewer data. Mutants that have similar decision boundaries could misclassify the exact data close to the boundaries. They both have a negative contribution to diversity. If the entropy is used to measure the various changes of decision boundaries, the above mutants could be found.

4 Empirical Study

We conduct an empirical study to verify the effectiveness of the proposed method. This experiment is conducted on Keras (ver.2.3.1) with Tensorflow (ver.1.15.2) backend, which runs on a high-performance computer with an Ubuntu system (ver.20.10) on I9-10900K CPU with 64 GB of RAM and an NVIDIA RTX3080 GPU with 10G.

4.1 Experimental Design

We select MNIST [6], which is frequently used in DL software testing research, as the experimental subject. MNIST is a ten-classes classification number-picture dataset. The number in it ranges from 0 to 9. MNIST contains 70,000 pictures, including 60,000 training pictures and 10,000 testing pictures. The distribution of each class of pictures are the same.

We select model$_1$ [19] and model$_2$ [20] as the evaluation subjects. Model$_1$, named LeNet-5, is a classic ConvNet model in the DL area. It performs well in solving classification problems such as handwriting recognition [21]. Currently, researchers have treated LeNet-5 as the benchmark model for measuring DL testing adequacy [22]. We also selected model$_2$ as a supplement because it is widely used in the DL mutation testing area [5]. Table 2 summarizes the characteristics of the models used in the experiments. For each model, its name (column 1), the number of trainable parameters (column 2), the number of convolutional layers (column 3), the number of pooling layers (column 4), and their classification accuracies (columns 5 and 6) are described. As is shown, both models achieve a high classification accuracy on the MINIST dataset.

Table 2. Evaluation subjects

Model	#Trainable parameters	#Convolutional layers	#Pooling layers	Training accuracy on MNIST	Testing accuracy on MNIST
model$_1$	107,786	2	2	99.14%	98.89%
model$_2$	694,402	4	2	98.45%	97.71%

Operator LR works on the Dense layer and BatchNormalization layer. The input shape and the output shape should be consistent. Since both model$_1$ and model$_2$ do not meet these requirements, LR cannot generate mutants. With regards to operators LD and LA$_m$, they work on the layer that has the consistent shape of input and output. Model$_1$ and Model$_2$ do not meet this requirement. Thus, LD and LA$_m$ also cannot be used. We discard them in our empirical study.

Usually, the threshold of defect detected rate on the test set is 20% [5]. Mutants that have a higher defect detected rate will be discarded. For each of the remaining thirteen operators, we randomly generate 30 mutants, where the detected rate of each is equal to or lower than 20%. Note that fewer AFR$_m$ mutants (i.e., one for model$_1$, two for model$_2$) exist. To summarize, as shown in Table 3, thirteen DL mutation operators and 723 mutants (i.e., 361 for model$_1$, 362 for model$_2$) are used in our empirical study.

Mutation operator reduction aims to decrease the number of mutants. Generating, compiling, and executing a mutant will cost a specific time. Thus, the more mutants, the more time we'll spend on mutation testing. Therefore, we use the reduction ratio (i.e., $|MUT_{reduced}|/|MUT_{all}|$) to evaluate the effectiveness of mutation operator reduction. Mutation operator reduction can also improve the accuracy of mutation testing. We use mutation score to evaluate the ability of accuracy improvement [5].

Table 3. Number of candidate mutants

Operators		MO_d	MO_p	MO_n	MO_l	Total
No. of mutants	model$_1$	150	60	150	1	361
	model$_2$	150	60	150	2	362

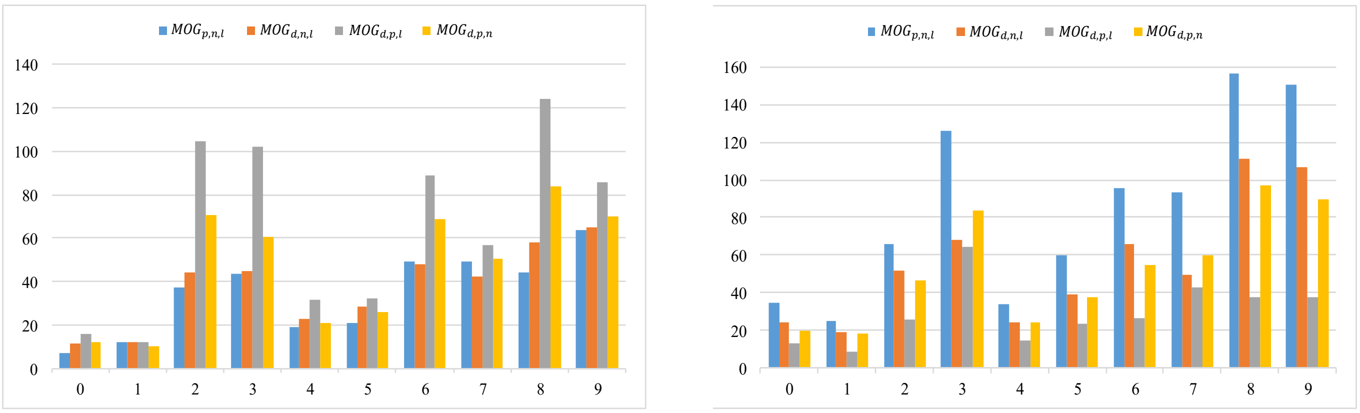

Figure 6. *DocEntropy* values of all classes

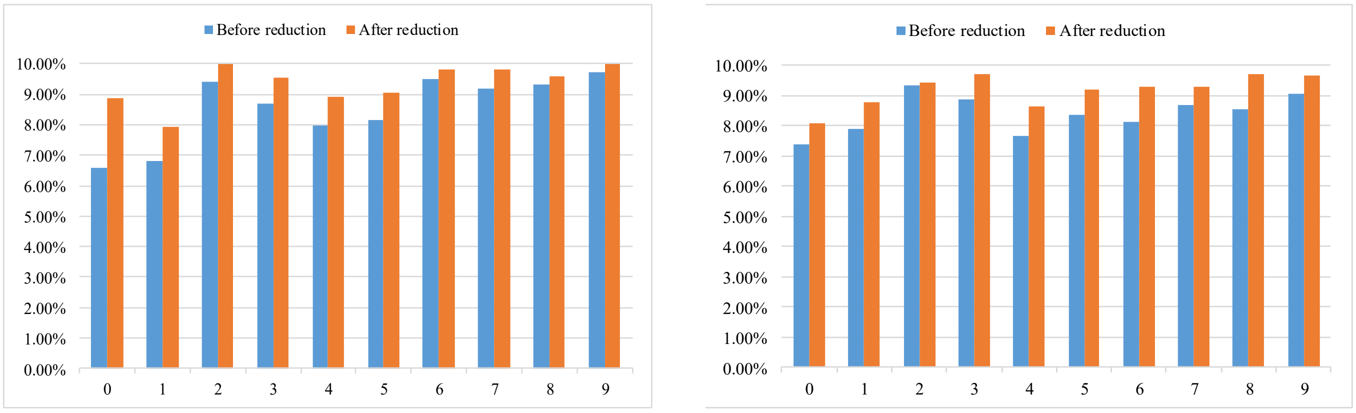

Figure 7. Mutation scores of all classes

Table 4. Result of operators and mutants reduction

Results	model$_1$			model$_2$		
	Before reduction	After reduction	Reduction radio	Before reduction	After reduction	Reduction radio
No. of operators	13	8	38.46%	13	8	38.46%
No. of mutants	361	240	33.52%	362	240	33.70%

4.2 Experimental Results and Analysis

Figure 6 depicts the *DocEntropy* values corresponding to the operator groups, $MOG_{p,n,l}$, $MOG_{d,n,l}$, $MOG_{d,p,l}$, and $MOG_{d,p,n}$. For each class of model$_1$, the mutants generated by $MOG_{d,p,l}$ archive the highest *DocEntropy* values, shown in Figure 6 (a). For example, in class 8, the value of $MOG_{d,p,l}$ is 124.07, which is an increase of nearly 47.39% compared to the second $MOG_{d,p,n}$, 84.18. The average value of $MOG_{d,p,l}$ is 65.51, and the second is 47.44 of $MOG_{d,p,n}$. It is an increase of nearly 38.09%. This shows that the change of decision boundaries of the mutants generated by $MOG_{d,p,l}$ is more diverse than the change of the other three groups. Compared with MO_d , MO_p , and MO_l , MO_n is not included in $MOG_{d,p,l}$, but is included in $MOG_{p,n,l}$, $MOG_{d,n,l}$, and $MOG_{d,p,n}$. The mutants generated by MO_n have a negative effect on the change of decision boundaries. Therefore, MO_n should be reduced in model$_1$. Operators in $MOG_{d,p,l}$ composed of MO_d, MO_p, and MO_l and should be selected as the reduced result.

In model$_1$, the mutation scores of mutants generated by $MOG_{d,p,l}$ are compared with ones generated by all mutation operators. The result is shown in Figure 7 (a). After MO_n are reduced, the mutation score of each class is improved. Among them, the most considerable improvement is class 0. The mutation score increases by nearly 34.86%, from 6.57% to 8.86%. In classes 2 and 9, the mutation score reaches the highest value of 10%. The average mutation score of all classes increases by nearly 9.61%. This shows that the mutants generated by the reduced mutation operator group can be killed by more data, which improves mutation score accuracy.

As shown in Table 4, the total number of mutation operators is reduced from thirteen to eight after selecting $MOG_{d,p,l}$, and the reduction ratio is 38.46%. The total number of mutants generated and executed was decreased from 361 to 240, and the reduction ratio was 33.52%.

For each class of Model$_2$, we perform a similar analysis. The difference is that the mutants generated by $MOG_{p,n,l}$ have the highest *DocEntropy* values, shown in Figure 6 (b). Take class 3 for an example. The value of $MOG_{p,n,l}$ is 125.94, which is an increase of nearly 50.59 % compared to the second $MOG_{d,p,n}$, 83.63. The average value of $MOG_{p,n,l}$ is 84.17, and the second is 55.97 of $MOG_{d,n,l}$. It is an increase of nearly 50.28%. This shows that the change of decision boundaries of mutants generated by $MOG_{p,n,l}$ is more diverse than the change of other three groups. Compared with $MO_p, MO_l, and\, MO_l, MO_d$ is not included in $MOG_{p,n,l}$, but is included in $MOG_{d,n,l}$, $MOG_{d,p,l}$, and $MOG_{d,p,n}$. The mutants generated by MO_d have a negative effect on the change of decision boundaries. Therefore, MO_d should be reduced. Operators in $MOG_{p,n,l}$ composed of MO_p, MO_n, and MO_l and should be selected as the reduced result.

In model$_2$, the mutation scores of mutants generated by $MOG_{p,n,l}$ are compared with ones of all mutants. The result is shown in Figure 7 (b). After MO_d are reduced, the mutation score of each class is improved. Among them, the most considerable improvement is class 8. The mutation score increased nearly 13.43%, from 8.56% to 9.71%. The average score of all classes increased by almost 9.29%, from 8.40% to 9.18%. This shows that the method improves the mutation score accuracy.

As is shown in Table 4, the total number of mutation operators is reduced from 13 to 8 after selecting $MOG_{p,n,l}$, and the reduction ratio is 38.46%. The total number of mutants generated and executed was decreased from 362 to 240, and the reduction ratio was 33.70%.

It is observed that the level of mutation operators reduced on model$_1$ and model$_2$ is different. Still, both can effectively select the operator set corresponding to the mutants with the most diverse decision boundary changes. The conclusion is that the mutation operator reduction method on model$_1$ and model$_2$ can effectively reduce the mutants, decrease the number of mutants generated and executed, and improve the accuracy of mutation scores.

4.3 Threats to Validity

This chapter mainly analyzes the threats from internal validity and external validity.

The internal validity mainly includes two aspects. First, the way the operator combination is worthy of analysis. One level of operators is ignored to form a group of the aggregation of multiple types of mutation operators. In the experiment, we can always find a mutation operator group with much high *DocEntropy* for all classes, proving the method has strong stability. On model$_1$, the mutation operators MO_d and MO_p, which act on the two related objects before training, produce higher quality mutants. With the increase of the model's trainable parameters, the operators MO_p and MO_n related to the model are more effective on model$_2$. The above two aspects show that the combination of mutation operators is effective and reduces the threat. Second, the reliability of some artificial setting parameters in the experiment is worthy of analysis. To avoid contingency, 30 mutants are generated for each effective mutation operator to make the results convincing. Moreover, the mutation rate was set to 1%, 5%, and 10% to generate more effective mutants. After comparing the number of non-valid mutants generated, we select 1% of the least non-valid mutants to increase overall quality. It effectively ensures the conduct of the experiment.

The external validity is mainly the effect of the equivalence [23] and redundancy [24], which are the objects that need to be considered in mutation testing. In the paper, decision boundary change measurement takes them into full consideration by *DocEntropy* calculation. Specifically, mutants with decision boundaries similar to the original model have fewer changed boundary distances to be measured, negatively contributing to *DocEntropy*. This shows that the reduction method proposed is sensitive to equivalences. In addition, mutants with similar decision boundaries can reduce the diversity of decision boundary changes. *DocEntropy* would rapidly decrease in the set of low-diversity mutants. This shows that the reduction method proposed is sensitive to redundancies. Thus, the reduction method proposed can reduce the mutation operators of the generated equivalences and redundancies.

5 Related Work

5.1 DL Software Testing

With DeepXplore [25] presented in 2018, the testing of DL software has gradually been valued by researchers. For a time, research directions such as test input generation, test coverage indicators, and test input selection have been developed one after another.

Test input generation is mainly used to generate data that the model can misjudge. The most used is the adversarial sample generation technology. They add interference to the data that is not visible to the human eye, causing errors in the model output [26]. Adversarial models are another effective way to test input generation. DeepTest [27] synthesizes the autopilot scene with various weather backgrounds and generates images that make the autopilot decision model errors. Test coverage indicators are such as neuron coverage [28]. These methods seek to find more model defects by increasing the coverage of the DL model. However, coverage indicators are still difficult to interpret [29]. The test input selection is mainly to improve the effectiveness of the test by selecting data in the dataset. DeepGini [15] uses a statistical method to identify data that can misclassify the model quickly. The quality of the model is improved by training these data. From the uncertainty of the model output, Ma et al. [30] selected data that could be classified incorrectly by the model. Experiments showed that this method is more efficient than test coverage indicators.

5.2 Mutation Testing Cost Reduction

Traditional program-based mutation testing is often divided into three steps: select mutation operators, generate mutations, and calculate mutation scores. According to whether a mutant is generated before reduction, mutation reduction is divided into mutation operator reduction and mutant reduction.

Since the reduction of mutation operators was proposed in 1991, it has been used quite a lot. Mathur. Offutt et al. [31] presented 2-selective, 4-selective, and 6-selective mutation strategies for the mothra mutation operator set, which ignores 2, 4, or 6 mutation operators to achieve the effect of reducing the number of mutants. Namin et al. [32] chose to use a statistical analysis program to identify the characteristic of mutation operators and achieve mutation operators reduction. Experimental results show that this method can fully predict the mutation score using a small number of mutants. Silva et al. [33] applied the search-based testing method to mutation testing, selecting a more efficient mutation operator and effectively reducing the cost of testing.

The reduction method of mutants, which is more regular, is mainly divided into three random selections, specific kinds of mutants selection, and multiple operators acting as one mutant. The random selection method selects mutants from the generated mutants according to a certain ratio, such as 10%, 20%. According to the experiment of Wong et al. [34], it can be concluded that when the ratio exceeds 10%, the random selection method is feasible. Zhang et al. [35] proved that randomly selecting mutants is better than mutants selection oriented by mutation operators. Meanwhile, this also shows that reduction based on mutants is more valuable for research. However, Yao et al. [36] reduced the mutants based on the characteristic of the mutation operators. Experimental results show that this method can effectively reduce equivalent mutants and make mutation scores more accurate. Harman et al. [37] found that injecting multiple mutation operators into a program can achieve mutant reduction. Simultaneously, by increasing the requirements for killing mutants, the set of test cases is increased. This also reflects that the method of multiple operators acting as one mutant is efficient.

6 Summary

In the paper, we firstly propose a mutation operator reduction method for DL mutation testing. Specifically, we calculate the distances of the decision boundaries between the original model and each mutant. Then, we apply *DocEntropy* to evaluate the diversity of distances, representing the decision boundary changes. Finally, we select the operators with the highest value as the reduction result. In the experiment, this method successfully reduced thirteen mutation operators to eight, decreasing nearly 33.61% mutants generated and executed. Moreover, the accuracy of mutation scores improves an average of 9.45%. In the future, we will try to use high-order mutants to improve the efficiency of DL software mutation testing.

Acknowledgements

The work is partly supported by the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX21_1140), the General Project of Basic Natural Science in Colleges and Universities of Jiangsu Province (21KJB520027), the Key Project of University Education Information Research (2021JSETKT023) and the Project of University-Industry Collaborative Education (202002180001).

References

[1] J. Li, J. Wang, X. Chen, Z. Luo, Z. Song, Multiple Task-driven Face Detection Based on Super-resolution Pyramid Network, *Journal of Internet Technology*, Vol. 20, No. 4, pp. 1263-1272, July, 2019.

[2] C. Chen, A. Seff, A. Kornhauser, J. Xiao, Deepdriving: Learning Affordance for Direct Perception in Autonomous Driving, *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, 2015, pp. 2722-2730.

[3] Z. Wang, M. Yan, S. Liu, J. Chen, D. Zhang, Z. Wu, X. Chen, Survey on Testing of Deep Neural Networks, *Journal of Software*, Vol. 31, No. 5, pp. 1255-1275, May, 2020.

[4] A. P. Mathur, W. E. Wong, A Theoretical Comparison between Mutation and Data Flow based Test Adequacy Criteria, *Proceedings of 1994 ACM Computer Science Conference (CSC'94)*, Phoenix, Arizona, USA, 1994, pp. 38-45.

[5] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. J. Xu, C. Xie, L. Li, Y. Liu, J. Zhao, Y. Wang, DeepMutation: Mutation Testing of Deep Learning Systems, *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, Memphis, TN, USA, 2018, pp. 100-111.

[6] L. Deng, The Mnist Database of Handwritten Digit Images for Machine Learning Research [Best of the Web], *IEEE Signal Processing Magazine*, Vol. 29, No. 6, pp. 141-142, November, 2012.

[7] F. Belli, C. J. Budnik, A. Hollmann, T. Tuglular, W. E. Wong, Model-based Mutation Testing—Approach and Case Studies, *Science of Computer Programming*, Vol. 120, pp. 25-48, May, 2016.

[8] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, M. Harman, Mutation Testing Advances: An Analysis

and Survey, *Advances in Computers*, Vol. 112, pp. 275-378, 2019.

[9] T.-G. Tsuei, C. H. Ting, H.-C. Chao, Laws of Computing: A View from Forth, *Journal of Internet Technology*, Vol. 1, No. 2, pp. 59-66, December, 2000.

[10] M. E. Delamaro, J. Offutt, P. Ammann, Designing Deletion Mutation Operators, *IEEE Seventh International Conference on Software Testing, Verification and Validation*, Cleveland, OH, USA, 2014, pp. 11-20.

[11] W. Shen, Y. Li, L. Chen, Y. Han, Y. Zhou, B. Xu, Multiple-boundary Clustering and Prioritization to Promote Neural Network Retraining, *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, VIC, Australia, 2020, pp. 410-422.

[12] H. Karimi, J. Tang, Decision Boundary of Deep Neural Networks: Challenges and Opportunities, *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM)*, Houston, TX, USA, 2020, pp. 919-920.

[13] W. Y. Chiu, G. G. Yen, T. K. Juan, Minimum Manhattan Distance Approach to Multiple Criteria Decision Making in Multiobjective Optimization Problems, *IEEE Transactions on Evolutionary Computation*, Vol. 20, No. 6, pp. 972-985, May, 2016.

[14] G. F. C. Contreras, H. J. Dulcé-Moreno, R. A. Melo, Arduino Data-logger and Artificial Neural Network to Data Analysis, *5th International Meeting for Researchers in Materials and Plasma Technology*, San José de Cúcuta, Colombia, 2019, pp. 1-7.

[15] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, Z. Chen, Deepgini: Prioritizing Massive Tests to Enhance the Robustness of Deep Neural Networks, *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, Virtual Event, USA, 2020, pp. 177-188.

[16] B. Kurtz, P. Ammann, J. Offutt, M. E. Delamaro, M. Kurtz, N. Gökçe, Analyzing the Validity of Selective Mutation with Dominator Mutants, *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, 2016, pp. 571-582.

[17] C. Ezeh, T. Ren, Y. Xu, S. Sun, Z. Li, Entropy and Structural-Hole Based Node Ranking Methods, *Journal of Internet Technology*, Vol. 22, No. 5, pp. 1011-1019, September, 2021.

[18] Q. Shi, Z. Chen, C. Fang, Y. Feng, B. Xu, Measuring the Diversity of a Test Set with Distance Entropy, *IEEE Transactions on Reliability*, Vol. 65, No. 1, pp. 19-27, March, 2016.

[19] Y. LeCun, LeNet-5, Convolutional Neural Networks, *URL:http://yann.lecun.com/exdb/lenet*.

[20] C. Xiao, B. Li, J. Zhu, W. He, M. Liu, D. Song, *Generating Adversarial Examples with Adversarial Networks*, January, 2018, https://arxiv.org/abs/1801.02610.

[21] A. Khan, A. Sohail, U. Zahoora, A. S. Qureshi, A Survey of the Recent Architectures of Deep Convolutional Neural Networks, *Artificial Intelligence Review*, Vol. 53, No. 8, pp. 5455-5516, December, 2020.

[22] S. Yan, G. Tao, X Liu, J. Zhai, S. Ma, L. Xu, X. Zhang, Correlations Between Deep Neural Network Model Coverage Criteria and Model Quality, *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, Virtual Event, USA, 2020, pp. 775-787.

[23] L. Madeyski, W. Orzeszyna, R. Torkar, M. Józala, Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation, *IEEE Transactions on Software Engineering*, Vol. 40, No. 1, pp. 23-42, January, 2014.

[24] C. Iida, S. Takada, Reducing Mutants with Mutant Killable Precondition, *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops*, Tokyo, Japan, 2017, pp. 128-133.

[25] K. Pei, Y. Cao, J. Yang, S. Jana, Deepxplore: Automated Whitebox Testing of Deep Learning Systems, *Proceedings of the 26th Symposium on Operating Systems Principles*, Shanghai, China, 2017, pp. 1-18.

[26] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and Harnessing Adversarial Examples, December, 2014, https://arxiv.org/abs/1412.6572.

[27] Y. Tian, K. Pei, S. Jana, B. Ray, Deeptest: Automated Testing of Deep-neural-network-driven Autonomous Cars, *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, Gothenburg, Sweden, 2018, pp. 303-314.

[28] L. Ma, F. J. Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, Y. Wang, Deepgauge: Multi-granularity Testing Criteria for Deep Learning Systems, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*, Montpellier, France, 2018, pp. 120-131.

[29] J. Chen, M. Yan, Z. Wang, Y. Kang, Z. Wu, Deep Neural Network Test Coverage: How Far Are We?, October, 2020, https://arxiv.org/abs/2010.04946.

[30] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, Y. L. Traon, Test Selection for Deep Learning Systems, *ACM Transactions on Software Engineering and Methodology*, Vol. 30, No. 2, pp. 1-22, April, 2021.

[31] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, C. Zapf, An Experimental Determination of Sufficient Mutant Operators, *ACM Transactions on Software Engineering and Methodology*, Vol. 5, No. 2, pp. 99-118, April, 1996.

[32] A. S. Namin, J. Andrews, D. Murdoch, Sufficient Mutation Operators for Measuring Test Effectiveness, *2018 ACM/IEEE 30th International Conference on Software Engineering (ICSE)*, Leipzig, Germany, 2008, pp. 351-360.

[33] R. A. Silva, S. R. S de Souza, P. S. L. Souza, A Systematic Review on Search Based Mutation Testing, *Information and Software Technology*, Vol. 81, pp. 19-35, January, 2017.

[34] W. E. Wong, A. P. Mathur, Reducing the Cost of Mutation Testing: An Empirical Study, *Journal of Systems and Software*, Vol. 31, No. 3, pp. 185-196, December, 1995.

[35] L. Zhang, S. S. Hou, J. J. Hu, T. Xie, H. Mei, Is Operator-based Mutant Selection Superior to Random Mutant Selection?, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, Cape Town, South Africa, 2010, pp. 435-444.

[36] X. Yao, M. Harman, Y. Jia, A Study of Equivalent and Stubborn Mutation Operators Using Human Analysis of Equivalence, *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, Hyderabad, India, 2014, pp. 919-930.

[37] M. Harman, Y. Jia, W. B. Langdon, A Manifesto for Higher Order Mutation Testing, *Third International Conference on Software Testing, Verification, and Validation Workshops*, Paris, France, 2010, pp. 80-89.

Biographies

Li-Chao Feng received his B.S. degree in computer science and technology from Nanjing Tech University, Jiangsu, China, in 2020. He is currently pursuing an M.S. degree in Nanjing Tech University, Jiangsu, China. His current research interest is intelligent software testing.

Xing-Ya Wang received his B.S. and Ph.D. degrees in computer science and technology from China University of Mining and Technology, Jiangsu, China, in 2012 and 2017. He is currently the Associate Professor at Nanjing Tech University. His current research interest includes AI software testing and smart contract testing.

Shi-Yu Zhang received his B.S. degree in computer science and technology from Nanjing Tech University, Jiangsu, China, in 2021. He is currently pursuing an M.S. degree in Nanjing Tech University, Jiangsu, China. His current research interest includes intelligent software testing, Blockchain (smart contract) analysis and testing.

Rui-Zhi Gao received his B.S. degree from Nanjing University and then received his M.S. degree and Ph.D. degree under the supervision of Professor W. Eric Wong at UTD. Dr. Gao focuses on software testing and program debugging. He is now working as a Principal Software Development Engineer at Sonos Inc.

Zhi-Hong Zhao received his B.S., M.S., and Ph.D. degrees in computer science and technology from Nanjing University, Jiangsu, China, from 1993 to 2002. He is currently the Professor in computer science and technology at Nanjing Tech University. His current research interest includes Software engineering, information system engineering, machine learning.