Evaluating the Robustness of Test Selection Methods for Deep Neural Networks

Qiang Hu*, Yuejun Guo†, Xiaofei Xie‡, Maxime Cordy*, Wei Ma§, Mike Papadakis* and Yves Le Traon*
*University of Luxembourg, Luxembourg
†Luxembourg Institute of Science and Technology, Luxembourg
‡Singapore Management University, Singapore
§Nanyang Technological University, Singapore

***Abstract*—Testing deep learning-based systems is crucial but challenging due to the required time and labor for labeling collected raw data. To alleviate the labeling effort, multiple test selection methods have been proposed where only a subset of test data needs to be labeled while satisfying testing requirements. However, we observe that such methods with reported promising results are only evaluated under simple scenarios, e.g., testing on original test data. This brings a question to us: are they always reliable? In this paper, we explore when and to what extent test selection methods fail for testing. Specifically, first, we identify potential pitfalls of 11 selection methods from top-tier venues based on their construction. Second, we conduct a study on five datasets with two model architectures per dataset to empirically confirm the existence of these pitfalls. Furthermore, we demonstrate how pitfalls can break the reliability of these methods. Concretely, methods for fault detection suffer from test data that are: 1) correctly classified but uncertain, or 2) misclassified but confident. Remarkably, the test relative coverage achieved by such methods drops by up to 86.85%. On the other hand, methods for performance estimation are sensitive to the choice of intermediate-layer output. The effectiveness of such methods can be even worse than random selection when using an inappropriate layer.**

I. Introduction

Deep learning (DL) has been a critical technique in our daily life and multiple DL-based systems have been deployed, for example, face recognition systems [1], chatbots [2], and self-driving cars [3]. Similar to conventional software systems that need to be systematically tested, testing DL-based systems is crucial to ensure that they meet the performance expectation and user requirements. Generally, the key component of DL-based systems is the deep neural network (DNN). Thus, the straightforward way of testing such systems is to evaluate their embedded pre-trained DNNs.

In a nutshell, DNN follows a data-driven paradigm and learns the inference logic from the training data, then makes decisions for new data. High-quality training data are important for preparing DNNs, and diverse test data with their corresponding labels are necessary for testing the performance of pre-trained DNNs. However, labeled test data are difficult to obtain due to the heavy labeling process, intensive human effort, and required domain knowledge. How to efficiently and fully test DNNs with fewer labeled test data is a challenging problem and becomes a hot research direction recently.

One promising solution to tackle this labeling problem is test selection for deep learning [4] – select and only label a subset of data from the entire test set. Multiple test selection methods have been proposed mainly by the software engineering community. Those methods can be roughly divided into two categories, 1) test selection for quick fault detection, and 2) test selection for model performance estimation. For simplicity, we call these two methods **fault detection method** and **performance estimation method**. Fault detection methods tend to identify the test data that have a higher probability to be misclassified by the model as faults. This kind of data can be further used for repairing the model (mainly by retraining) to enhance its performance. Performance estimation methods try to select a small size of data that can represent the entire test set. The model performance on the entire test set can be then approximated by using this subset. Here, robust means the capability of test selection methods to preserve their effectiveness on unusual (often, altered) data.

Similar to DL targeted test generation techniques whose reliability needs to be carefully studied [5] before real usage. The reliability of test selection methods also requires systematic exploration to check when they could fail, and more importantly, how harmful the failure could bring to the users. Otherwise, under the risky test selection methods, the testing requirements might not be satisfied and the labeling budget will be wasted. For example, fault detection methods rely on the assumption – there is a strong correlation between the probability of misclassification and the uncertainty of the data. They believe the data with high uncertainty (i.e., the model has low confidence in the data) should be misclassified by the model. Here, uncertainty is the proneness of input data to be mishandled by a model, which can be measured in different ways, e.g., entropy of the output probabilities. Confidence refers to how much the model believes it has made the correct decision. In classification models, this is the class probability as output by the softmax layer. However, this assumption leads the selection process to ignore faults with low uncertainty. Thus, when facing a large number of low uncertain data, the effectiveness of fault detection methods may degrade a lot. We conjecture that *existing test selection methods are not robust.* It is necessary to know what are their limitations for reliable usage.

In this paper, we study the above problems and focus on exploring when and to what extent the test selection methods fail. Concretely, first, we collect and make a short survey of existing

fault detection methods and performance estimation methods. In total, there are eight methods for fault detection and three methods for performance estimation (including random selection). Second, we analyze the potential pitfalls that could make these methods fail based on their design strategies. After that, we conduct an empirical study to reveal whether these issues exist and if yes, how harmful they are. Here, we design test generation methods to trigger these issues. Finally, based on our findings, we provide guidelines on how to robustly evaluate test selection methods when developing new ones. In total, our study covers five widely studied datasets (e.g., Traffic-sign, CIFAR100) with two popular model architectures (e.g., ResNet, DenseNet) for each dataset. Based on our study, we found:

- All fault detection methods are fooled by two types of test data, 1) correctly classified but with high uncertainty data, and 2) wrongly classified but with low uncertainty data. On average, the effectiveness of test selection methods drops 52.49% and 39.80% when facing these two types of test data, respectively.
- These two types of test data have a huge negative impact on the selection-guided model repair (i.e. retraining). When facing them, even random selection can achieve better repair results than well-designed methods like PRIMA, TestRank, and DSA. Specifically, 55 out of 80 repaired models have accuracy degradation when dealing with correctly classified but uncertain data.
- The effectiveness of existing performance estimation methods highly depends on the choice of output from the intermediate layer. They can only outperform the random selection with handpicked layers.

To summarize, the main contributions of this paper are:

1) This is the first work that explores the limitations of test selection methods for DNNs.
2) We reveal that existing 1) fault detection methods suffer from the correctly (wrongly) classified but uncertain (with high confidence) test data, and 2) performance estimation methods are sensitive to the choice of intermediate layers.

II. Background and Related Work

A. Deep Learning Testing

Roughly speaking, the objectives of testing DNNs mainly lie in two parts, 1) testing the natural robustness of DNNs [6]–[8] for ensuring their performance facing data with diverse distributions (including the same distribution as training data), and 2) testing the adversarial robustness of DNNs [9], [10] for potential security issues (e.g., adversarial attack). For the natural robustness testing, developers need to prepare as many as possible test data that follow different data distributions, and then assess DNN models accordingly. For adversarial robustness testing, generating adversarial examples and using them to test the models is a common way. Utilizing adversarial robustness certification techniques [11], [12] to strictly verify the model robustness is another way. For both types of DNN testing, labeled test data are necessarily needed. In this paper, we focus on the first objective of DNN testing and target the test data selection for efficient deep learning testing.

B. Test Selection for DNNs

Test selection methods are proposed for efficiently testing DNNs without heavy labeling effort. There are two types of test selection methods, the first one tries to select data that are most likely misclassified by the model [4], [13]–[18], and the second selects data [19], [20] to approximate the model performance on the entire test set. We introduce methods from each type in Section III. In this paper, instead of proposing new test selection methods, we reveal and study when existing test selection methods could fail.

C. Empirical study on Deep Learning Testing

Similar to this paper, many works [21] conducted empirical studies to investigate the usefulness of existing deep learning testing techniques. Yan *et al.* [22] and Dong *et al.* [23] studied the correlations between the neural coverage [24], [25] criteria (a type of deep learning testing criteria) and the model quality (mainly focus on the adversarial robustness of models). Jahangirova *et al.* [26] empirically evaluated the effectiveness of mutation testing for DNNs by using different mutation operators. Hu *et al.* [27] specifically studied how the test selection methods guided model retraining performance when facing data with different types of distribution shifts. Two works [4], [28] studied the effectiveness of fault detection target test selection methods, and they found that simple methods perform well in terms of fault detection and model retraining. Different from existing studies, our work focuses on two types of test selection methods and does not only consider simple test data. We build more complex testing scenarios to challenge the test selection methods and reveal their weak point.

III. Test Selection Methods Analysis

In total, We collect 10 representative test selection methods plus the random selection based on existing works [4], [28] and our review. Table I presents a brief description of methods, data features used for conducting selection, target tasks, and the venue where a method comes from. According to the purpose of use, existing test selection methods can be classified into two types. The first one is test selection for fault detection (fault detection method), and the second is test selection for model performance estimation (performance estimation method). In this work, we focus on the classification task since most of the methods (7 out of 10) are specifically designed for this task.

Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ be a trained DNN model that maps input data $\mathcal{X}$ into the target space $\mathcal{Y}$. $x \in \mathcal{X}$ and y denote a given test sample and its true label, respectively. The model f assigns the label $\widehat{y}$ to x based on its learned knowledge from the training set. Let $p_{y_i}(x)$ be the likelihood of x belonging to class $y_i \in \mathcal{Y}$, $\widehat{y} = y_i$ if p_{y_i} is the maximum over all possible classes in $\mathcal{Y}$.

TABLE I
SUMMARY OF TEST SELECTION METHODS.

Type	Method Name	Feature Used	Target Task	Venue
Fault detection	Distance-based surprise adequacy (DSA) [13]	Intermediate output	Classification Regression	ICSE 2019
	DeepGini [14]	Final output	Classification	ISSTA 2020
	Multiple-boundary clustering and prioritization (MCP) [15]	Final output	Classification	ASE 2020
	Monte Carlo dropout uncertainty (MC) [4]	Final output	Classification	TOSEM
	Maximum probability (MaxP) [4]	Final output	Classification	TOSEM
	TestRank [16]	Logits output + graph model	Classification	NeurIPS 2021
	PRIMA [17]	Final output + ranking model	Classification Regression	ICSE 2021
	Adaptive test selection (ATS) [18]	Final output	Classification	ICSE 2022
Performance estimation	Cross entropy-based sampling (CES) [19]	Intermediate output	Classification	ESEC/FSE 2019
	Practical accuracy estimation (PACE) [20]	Intermediate output + clustering model	Classification Regression	TOSEM

A. Fault Detection Methods

In this paper, *fault* refers to perceptible discordance between the model decision and the expected outcome. In classification tasks that we study in this work, a fault is revealed by an incorrect label. Given a set of unlabeled test data and a DNN model, fault detection methods identify test data (faults) that are likely to be wrongly predicted by the model. Generally, compared to the correctly predicted data, people are more interested in fault data that are useful to analyze the weak/blind point of the DNN model and further enhance the model. In practice, the found faults are usually utilized as a *patch* to repair (via model retraining) DNN models.

Distance-based surprise adequacy (DSA) [13] is the very first method proposed to select test data. The main idea of DSA is to measure the difference in activation traces between a given test sample and the training set.

$$DSA\left(x\right) = \frac{dist_a}{dist_b} \tag{1}$$

where $dist_a$ is the Euclidean distance between x and its closest neighbor from the same class $\widehat{y}$ and $dist_b$ refers to the neighbor from a different class. A sample with a high difference can better reveal faults.

DeepGini [14] quantifies the uncertainty of x for f:

$$Gini\left(x\right) = 1 - \sum_{y_i \in \mathcal{Y}} p_{y_i}\left(x\right) \tag{2}$$

Test samples with high uncertainties are considered as insufficiently learned by f and selected to detect faults.

Multiple-boundary clustering and prioritization (MCP) [15] first performs the so-called boundary area clustering that divides data into different boundary areas (confusion areas of every two classes). Based on the prediction, MCP assigns a priority:

$$MCP\left(x\right) = \frac{p_{\widehat{y}}}{p_{y_s}} \tag{3}$$

where y_s is the predicted second most likely class. Next, from each boundary area, MCP evenly selects data with the minimum priority.

Monte Carlo dropout uncertainty (MC) [4] takes advantage of the Monte Carlo dropout technique f to obtain multiple predictions and then calculate the uncertainty score of the data. There are several variants of MC, we consider the famous one [29], [30] in which the uncertainty is calculated by:

$$MC\left(x\right) = 1 - \frac{\mid \{i \mid \widehat{y}_i = mode\left\{\widehat{y}_j, 1 \leq j \leq N\right\}\} \mid}{N} \tag{4}$$

where N is the number of applying the dropout to obtain predictions. $mode\left(.\right)$ is the function of identifying the predicted class that appears most often.

Maximum probability (MaxP) [4] considers $p_{\widehat{y}}\left(x\right)$ as the confidence level f is given x. Accordingly, data with low confidence are supposed to be useful to uncover faults.

TestRank [16] is a learning-based method that contains three main steps. First, it extracts two types of features from the input data, 1) the output from the logits layer as intrinsic attributes, 2) the graph information that contains the distance of this data (cosine distance) to others, and the label of this data. Then, a GNN model is built to learn the graph information and predict the learned contextual attributes of the data. Finally, the contextual attributes and intrinsic attributes are combined and fed to a simple binary classification model to learn the failure-revealing capability.

PRIMA [17] is another learning-based method that first generates mutants from both input data and models. Then, it extracts features of each input data from those mutants, e.g., the number of mutants that have different predicted labels from the original model on this data. Finally, PRIMA trains a ranking model (XGBoost ranking algorithm) based on the features of correctly and wrongly predicted data to identify if the unlabeled data is a fault.

Adaptive test selection (ATS) [18] is a method that totally depends on the final output probability of the data. Roughly speaking, it first projects the output vectors (built by the top-3 maximum vectors) to a plane and then calculates the coverage of each data on the plane. After that, the difference between the coverage of a single data and the whole test set is utilized as an identifier to distinguish the faults and normal test data.

B. Performance Estimation Methods

Given a set of unlabeled test data and a DNN model, performance estimation methods tend to select a representative subset from the entire test data. The model performance on this subset should be close to that on the entire test data. In this way, developers can quickly observe how the model performs on unseen data and then take the next step. For example, if the performance is lower than the exception, developers need to repair the model.

Cross Entropy-based Sampling (CES) [19] selects a subset of test data that have the minimum cross entropy with the entire set.

Practical accuracy estimation (PACE) [20] first clusters data into groups based on the hierarchical density-based spatial clustering of applications with noise algorithm. Next, from each group, PACE proportionally selects data to represent the entire group.

Note that, prediction outputs from intermediate layers of DNNs are required to calculate the cross entropy in CES and to conduct clustering in PACE.

C. Pitfalls of Test Selection Methods

We conjecture that there are three types of pitfalls that hinder the success of test selection methods.

Blind in high uncertainty but correctly predicted data. All uncertainty-based fault detection methods (DeepGini, MCP, MC, MaxP, PRIMA, and ATS) assume that data with high uncertainty are more likely to be misclassified by the model. However, this assumption only stands when the model is well-trained and has low bias. DSA and TestRank do not rely on this assumption. They utilize the *difference* between correctly predicted data and wrongly predicted data to determine the new faults. However, in the original test data, there are not many high-uncertainty but correctly predicted faults which makes the learned *difference* can not be generalized to this type of fault. We conjecture that all the fault detection methods will identify the high uncertainty but correctly predicted data as faults. And in this paper, we call this type of data as ***Type1*** data. The methods involved are DSA, DeepGini, MCP, MC, MaxP, TestRank, PRIMA, and ATS.

Blind in low uncertainty but wrongly predicted data. The same reason as the last point, we conjecture that all the fault detection methods cannot detect the low uncertainty but wrongly predicted fault data, and we call this type of data as ***Type2*** data. The methods involved are DSA, DeepGini, MCP, MC, MaxP, TestRank, PRIMA, and ATS.

Susceptible to layer selection. We found that all the existing model performance estimation methods rely on the intermediate output of the model. However, a DNN model normally consists of multiple hidden layers. We doubt that the effectiveness of such test selection methods is highly impacted by the choice of outputs of hidden layers, and it is difficult to make a clear conclusion about which layer is better across different datasets and models. The methods involved are DSA, CES, and PACE.

IV. Overview

In this section, we design a study to investigate if the potential non-robust features exist. Although a perfect test selection method may not exist, people should know what they need to pay attention to when proposing, evaluating, and applying test selection methods.

A. Study Design

Figure 1 illustrates the overview workflow of our empirical study. Overall, our plan is to explore and answer the following five research questions:

RQ1: *How do fault detection methods compare to each other?*
RQ2: *Can existing fault detection methods bypass uncertain but correctly classified data?*
RQ3: *Can fault detection methods identify high confidence but wrongly classified faults?*
RQ4: *How do uncertain (high confidence) but correctly (wrongly) classified data affect test selection-based model repair?*
RQ5: *How does the choice of intermediate layers affect performance estimation methods?*

Before studying the problems listed in Sec III-C, we first investigate the common concern of using test selection methods – do we have the best choice among the massive number of methods? To do this, we have a quick look at all fault detection methods and check, when we only consider the original test data, if there is a recommended method that performs consistently better than others, or if there are methods that have big performance variances across different datasets that are not recommended to use.

Then, we go deeper to study the robustness of test selection methods. For fault detection, as discussed in Section III-C, there are two types of test data that are difficult for existing test selection methods to handle, *Type1*: high uncertainty but correctly predicted data, and *Type2*: low uncertainty but wrongly predicted data. The first step of our study is to generate these two types of test data. We follow the previous work [31] and design a genetic algorithm (GA) based test generation technique. The details will be introduced in the next section. After the data preparation, we inject the generated test data into the original test data and obtain three groups of test data, 1) only original test data, 2) *Type1* test data + original test data, and 3) *Type2* test data + original test data. Then, we perform test selection on these three groups of test data and analyze the effectiveness of each method. In addition, since the final target of fault detection is to fix or repair the pre-trained model to make it bypass these faults, we conduct a study to utilize the selected test data to retrain (the common model repair approach) the model. Specifically, we first evenly split the three groups of test data into two parts, the candidate set and the new test set. Then, we perform test selection on the candidate set and combine the selected data with training data to retain the model with a few epochs. After that, we test the retrained model using the new test set and analyze the effectiveness of test selection methods when facing *Type1* and *Type2* test data in terms of model repair.

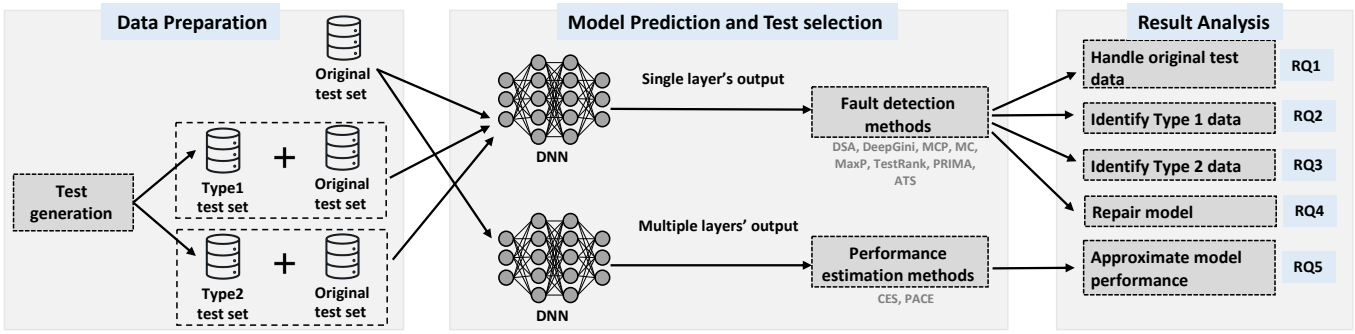


Fig. 1. Overview of the study design.

On the other hand, for the test selection-based model performance estimation, we choose outputs from different hidden layers gained by using the original test data and then fed them to the test selection methods. We analyze the difference in the estimated model performance to check the impact of the choice of the intermediate outputs.

B. Test Data Preparation

Test generation is a common practice to prepare test data when testing DNNs. In which, genetic algorithm [31], [32] based test generation technique is effective in generating diverse test data. To prepare the two types of test data, we design a very simple and flexible (easy to extend) GA-based test generation technique, as presented in Algorithm 1. Mention that, our purpose is to reveal the limitations of test selection methods instead of attacking them. Other methods like modified white or black-box adversarial attacks can also generate such kinds of test data. Investigation of them is left as future work.

Here, we explain the flexible components in the generation process that control whether to generate *Type1* or *Type2* test data. We use the notations defined in Section III. Given a DNN model f, a test data x and its true label y, $p_{\widehat{y}}(x)$ represents the probability of x belonging to the predicted class $\widehat{y}$ by f. y_s is the predicted second most likely class.

Seed preparation. We follow a previous work [33] to randomly select seed data from each class of the datasets to increase the diversity of the seeds.

Condition controls if *Type1* or *Type2* test data are successfully generated. For *Type1* test data, the generated data should fit all these three conditions: 1) $\widehat{y} = y$, 2) $p_{\widehat{y}}(x) \leq T_1$, and 3) $p_{\widehat{y}}(x) - p_{y_s}(x) \leq T_2$. For *Type2* test data, the generated data should fit all these two conditions: 1) $\widehat{y} \neq y$, 2) $p_{\widehat{y}}(x) > T_1$. Here, T_1 and T_2 are two predefined thresholds (see Table III).
Fitness function controls the evolution direction during test generation. For *Type1* data generation, the fitness function is the combination of $-(p_{\widehat{y}}(x) - p_{y_s}(x))$ and $-p_{\widehat{y}}(x)$. For *Type2*, the fitness function is $p_{y_s}(x)$ if the data are correctly classified, otherwise $p_{\widehat{y}}(x)$.

Crossover follows the work [31] and utilizes the tournament selection strategy to select two tournaments, and then chooses one data with the biggest fitness score from each tournament respectively to do randomly pixel changing.

Mutation aims to increase the diversity of the population. Here, we utilize image transformation techniques to generate mutants and control the perturbation size to ensure the semantics of the generated image do not change.

Algorithm 1: GA-based test generation

Input : $seed$: seed data
pop_size: size of population
$max\_iteration$: maximum number of iteration
$tour\_size$: size of tour data
Output : $X\_generated$: generated test data

$pop = Population\_Initialization(seed,\ pop\_size)$
$count\_num = 0$
while $count\_num < max\_iteration$ **do**
  for $X\_generated\ in\ pop$ **do**
    if $X\_generated\ fits\ Condition$ **then**
      return $X\_generated$;
    end
  end
  $new\_pop = []$
  $fitness = Fitness\_Calculation()$
  $individual = Select\_Best(pop,\ fitness)$
  $pop = pop \setminus individual$
  $new\_pop.update(individual)$
  $pop = Crossover(pop,\ tour\_size)$
  $pop = Mutation(pop)$
  $new\_pop.update(pop)$
  $pop = new\_pop$
end
$X\_generated = pop[0]$
return $X\_generated$;

C. Experimental Setup

Dataset and model. Table II lists the details of our studied datasets and models. MNIST [34] and SVHN [35] contain digital numbers. CIFAR10 [36] is a collection of color images with 10 categories (e.g., airplane, bird). CIFAR100 [34] is a more challenging version of CIFAR10 with fine-grained 100 categories (e.g., aquarium fish, flatfish). Traffic-Sign [37] contains traffic sign images and is commonly used for self-driving cars. For each dataset, we build two popular models

that are mainly from the LeNet [38], ResNet [39], VGG [40], and DenseNet [41] families.

TABLE II
DATASET AND MODEL.

Dataset	Class Number	Test Size	Model	Test Accuracy (%)
MNIST	10	10000	LeNet1	98.07
			LeNet5	98.87
SVHN	10	10000	LeNet5	87.55
			ResNet20	95.85
CIFAR10	10	10000	ResNet20	87.44
			VGG16	91.39
Traffic-Sign	43	12630	LeNet5	83.37
			VGG16	93.08
CIFAR100	100	10000	ResNet50	76.89
			DenseNet121	71.72

Configurations of test generation. There are some hyper-parameters in the test generation process. Investigating the best configuration is not our focus. Instead, we give recommended configurations used in our study that can already achieve our purpose. Table III lists the detailed configurations. And for the image mutation, we employ four image transformation techniques, image contrast changing, image brightness changing, blur noise adding, and Gaussian noise adding [6]. It is flexible and easy to extend with more mutation operators. To preserve the semantics of the generated data, we follow the work [42] and limit the maximum L-infinite perturbation size of the injected noise.

TABLE III
CONFIGURATIONS OF TEST GENERATION.

	Type1		Type2	perturbation	pop	max	tour
	T_1	T_2	T_1	size	size	iteration	size
MNIST	0.5	0.01	0.95	0.5	200	200	20
SVHN, CIFAR10	0.5	0.01	0.95	0.05			
Traffic	0.3	0.05	0.95	0.05			
CIFAR100	0.1	0.05	0.5	0.05			

Configurations of test selection. For the fault detection, we set the maximum labeling budget as 10% of the entire test data, which is a common setting in previous works [15], [18], [27]. For the performance estimation, we follow previous works [19], [20] and set the labeling budgets from 50 to 180 in intervals of 10. For the intermediate output selection for the performance estimation, we chose the last 1, 2, and 3 hidden layers in our study.

Evaluation methods. To evaluate the effectiveness of fault detection, we adopt the measurement, Test Relative Coverage (TRC), from the literature [16].

TRC is defined as:

$$TRC\left(X\right) = \frac{|F\_detected|}{\min\left(|F\_X|, Budgets\right)} \tag{5}$$

where $Budgets$ is the size of selected data.

Besides, we employ Student's t-test [43] which is a famous statistical analysis method to analyze the significance of the impact from the *Type1* and *Type2* data. For model repair, we use the absolute accuracy difference between the original model and the repaired model to quantify the effectiveness of related test selection methods.

Implementation. The main framework uses TensorFlow [44] 2.3.0. The implementation of each test selection method is modified from the source code provided by the original paper to fit our experiment environment. All experiments run on a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. We repeat the experiments with randomness factors five times and report the average results, e.g., test selection using PRIMA, the model repair process.

V. RESULTS

A. RQ1: Performance on Original Test Data

First, for the practical usage of test selection methods, we need to know if there is one that outperforms others and can be recommended to use. Figure 2 depicts the test relative coverage of each method. From the results, we can see the TRC of each method variant from different datasets and models, which means that when facing new datasets, it is hard to choose which one we should use. More specifically, Table IV shows the TRC of each method when the labeling budget is 10%. From the ranking of each method, we can draw the same conclusion as before that no method can always stand out. However, we found two methods, DeepGini and MaxP, have top-3 TRC scores in all situations. This finding is similar to the recent study [28] which reveals that simple methods work better for DL-targeted test selection.

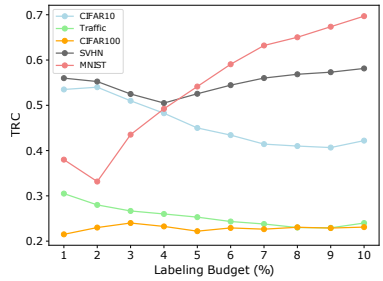
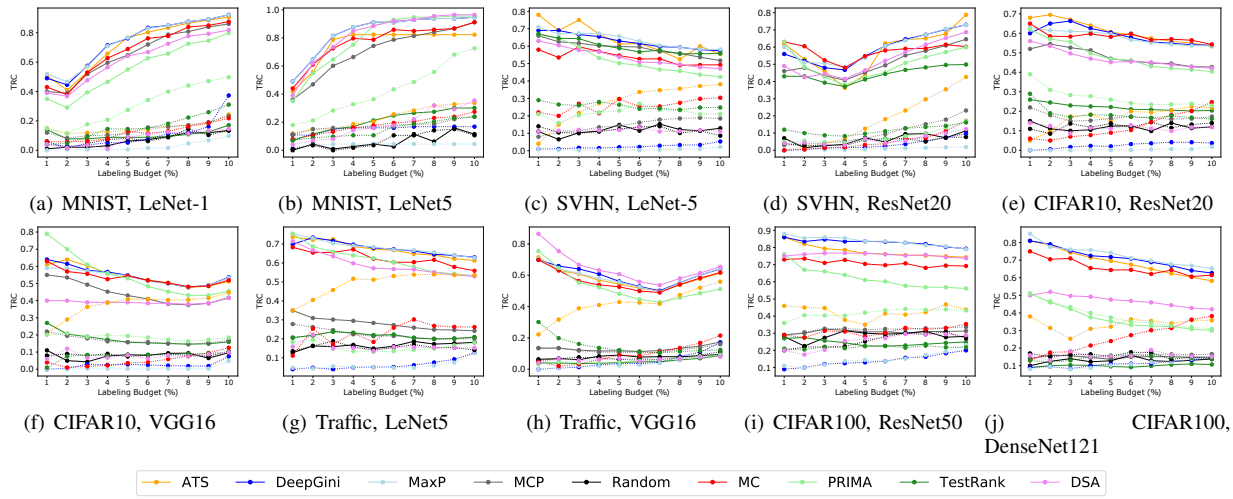


Fig. 3. TRC of MCP on different datasets.

TABLE IV
FAULT DETECTION PERFORMANCE OF EACH METHOD WITH LABELING BUDGET OF 10% – TRC(RANKING).

	Random	DSA	DeepGini	MCP	MC	MaxP	TestRank	PRIMA	ATS
MNIST-LeNet1	0.13(9)	0.82(6)	0.92(1)	0.86(5)	0.88(4)	0.92(1)	0.17(8)	0.80(7)	0.91(3)
MNIST-LeNet5	0.12(9)	0.96(1)	0.95(3)	0.91(5)	0.91(5)	0.95(3)	0.30(8)	0.96(1)	0.82(7)
SVHN-LeNet5	0.13(9)	0.47(7)	0.57(2)	0.52(5)	0.49(6)	0.58(1)	0.56(3)	0.42(8)	0.56(3)
SVHN-ResNet20	0.11(9)	0.69(4)	0.73(2)	0.65(5)	0.60(6)	0.73(2)	0.50(8)	0.60(6)	0.79(1)
CIFAR10-ResNet20	0.12(9)	0.42(6)	0.54(1)	0.43(5)	0.54(1)	0.54(1)	0.20(8)	0.40(7)	0.53(4)
CIFAR10-VGG16	0.10(9)	0.41(7)	0.54(1)	0.42(6)	0.52(3)	0.53(2)	0.16(8)	0.46(5)	0.51(4)
Traffic-LeNet5	0.18(9)	0.53(5)	0.63(1)	0.24(7)	0.56(4)	0.63(1)	0.21(8)	0.53(5)	0.61(3)
Traffic-VGG16	0.09(8)	0.65(1)	0.64(2)	0.17(7)	0.62(4)	0.64(2)	0.09(8)	0.51(6)	0.61(5)
CIFAR100-ResNet50	0.28(8)	0.74(3)	0.79(1)	0.31(7)	0.69(5)	0.79(1)	0.25(9)	0.56(6)	0.74(3)
CIFAR100-DenseNet101	0.14(8)	0.42(5)	0.63(2)	0.15(7)	0.62(3)	0.65(1)	0.11(9)	0.30(6)	0.58(4)
Average	0.14(9)	0.61(5)	0.69(2)	0.47(7)	0.64(4)	0.70(1)	0.25(8)	0.55(6)	0.67(3)
Variance	0.00(1)	0.04(7)	0.02(2)	**0.07(9)**	0.02(2)	0.02(2)	0.02(2)	0.04(7)	0.02(2)

Interestingly, besides proving the literature findings, we found that the TRC of MCP has a bigger variance across different datasets than others. When checking the TRC of MCP of each dataset, we can see MCP has the same level of performance as DeepGini and MaxP (e.g., 0.91 vs. 0.95 vs. 0.95) in datasets MNIST, SVHN, and CIFAR10. But when checking the results on Traffic and CIFAR100, the performance of MCP has a great degradation, and the difference between MCP, DeepGini, and MaxP becomes not negligible (e.g., 0.15 vs. 0.63 vs. 0.65). The reason for this unstable performance of MCP is that MCP is highly dependent on the number of classes of the dataset. MCP first divides data into fine-grained boundary areas before selecting. For a 10-class dataset (e.g., MNIST), the number of decision boundary areas is $A_{10}^2 = 90$. However, when the class number increases to



(a) MNIST, LeNet-1 (b) MNIST, LeNet5 (c) SVHN, LeNet-5 (d) SVHN, ResNet20 (e) CIFAR10, ResNet20

(f) CIFAR10, VGG16 (g) Traffic, LeNet5 (h) Traffic, VGG16 (i) CIFAR100, ResNet50 (j) CIFAR100, DenseNet121

Fig. 2. Test Relative Coverage (TRC) of Test Selection Methods on Original and *Type1* Test Data. Solid lines: original test data, dash lines: *Type1* test data.

100 (e.g., CIFAR100), the number of boundary areas also increases to $A^2_{100} = 9900$, which can easily exceed the labeling budget. In this case, MCP becomes a random-like selection method. Figure 3, the detailed TRC scores of MCP on different datasets, also confirm our analysis. MCP performs much worse on Traffic and CIFAR100 than on others.

Then, we go further and check the efficiency of each method. Table V presents the executing time of each test selection method when it ranks all the test data once. It is reasonable that the time cost increases along with the increasing complexity of datasets and model architectures. However, we found that, in the case of (SVHN-LeNet5 vs Traffic-LeNet5) and (CIFAR10-VGG16 vs Traffic-VGG16), only ATS and PRIMA have a big difference in the time cost. PRIMA has complex steps including preparing the mutants of data and models, and the total time cost is unstable and hard to analyze. For the ATS, the time cost increases significantly because 1) it tries to identify the faults from each class one by one, and 2) when the number of classes increases, the types of fault pattern (combination of classes) increase non-linearly, e.g., from A^3_{10} to A^3_{100}.

TABLE V
RUNNING TIME (SECONDS) OF FAULT DETECTION TARGET TEST SELECTION METHODS WHEN RANKING ALL THE TEST DATA ONCE.

	Random	DSA	DeepGini	MCP	MC	MaxP	TestRank	PRIMA	ATS
MNIST-LeNet1	0.44	10.26	11.50	0.61	1.11	0.45	30.90	185.38	18.92
MNIST-LeNet5	0.48	11.98	12.58	0.64	1.20	0.49	32.21	289.33	3.17
SVHN-LeNet5	0.77	12.06	18.57	0.93	1.74	0.79	27.86	286.37	2.06
SVHN-ResNet20	1.56	51.92	39.66	1.81	3.14	1.53	43.38	2085.41	19.08
CIFAR10-ResNet20	1.59	51.02	40.38	1.77	3.35	1.57	54.38	1980.69	31.24
CIFAR10-VGG16	1.17	131.21	27.06	1.31	2.54	1.19	76.70	3986.26	18.34
Traffic-LeNet5	0.93	14.09	22.24	1.08	2.14	0.96	35.94	2568.60	1035.86
Traffic-VGG16	1.37	101.23	31.44	1.52	3.01	1.38	76.40	2658.98	1475.31
CIFAR100-renset50	2.66	62.42	58.82	2.85	5.53	2.66	60.83	3183.44	5080.69
CIFAR100-denset101	4.82	58.49	102.55	5.00	9.76	4.76	56.70	3042.73	4170.33

Answer to RQ1: No methods have consistently better performance than others. MCP and ATS have significant effectiveness and efficiency drops, respectively, when handling data with a large number of classes.

B. RQ2: Impact of Type1 Test Data

To study how the test selection methods deal with the data that are correctly classified but with high prediction uncertainty. We generate *Type1* test data and inject it into the original test set, then evaluate the test selection methods accordingly. Here, the number of injected data is the same as the budget of selected (labeled) data.

Figure 2 presents the results of the 10 fault detection methods with the labeling budget ranging from 1% to 10%. In general, there is a clear gap between the results of the original test data and the *Type1* test data. All the test selection methods perform worse when the *Type1* data exists, which means that methods tend to select *Type1* data as the faults but in fact, they are not. Then, comparing each method, PRIMA and ATS are more effective than others when dealing with *Type1* test data, e.g., in MNIST, PRIMA is significantly better than other methods, and in SVHN, Traffic, and CIFAR100 datasets, ATS is better. The potential reason is that PRIMA mutates the data multiple times and then calculates the uncertainty scores. Although the generated test data are close to the decision boundary, their predictions may not change after mutation, thus, PRIMA does not identify these *Type1* test data as faults. ATS selects faults from diverse fault patterns and does not only select uncertain data. Besides, we statically compare the fault detection performance of each method on original test data and *Type1* data using t-test. The results show that except for *Random*, *TestRank*, and *MCP*, all the methods perform significantly worse on *Type1* data than on original test data (with a p-value <0.05).

More specifically, Table VI presents the average test relative coverage values of each test selection method among all the datasets and models when the labeling budget is 10%. We can see that except for random selection, only TestRank has a small performance degradation on the *Type1* test data, but its TRC on the original test data is not high. Other test selection methods have at least 42% test relative coverage drops, where

TABLE VI
AVERAGE TRC VALUES OF TEST SELECTION METHODS OVER ALL DATASETS AND MODELS (LABELING BUDGET 10%).

	Random	DSA	DeepGini	MCP	MC	MaxP	TestRank	PRIMA	ATS	Average
Ori	0.14	0.61	0.69	0.47	0.64	0.70	0.25	0.55	0.67	0.53
Type1	0.13	0.16	0.14	0.21	0.26	0.09	0.19	0.30	0.39	0.21
Diff	6.42% ↓	73.47% ↓	79.25% ↓	55.05% ↓	60.31% ↓	86.85% ↓	24.96% ↓	45.25% ↓	40.88% ↓	52.49% ↓

MaxP and DeepGini drop the greatest.

Answer to RQ2: Existing fault detection methods cannot distinguish real faults and *Type1* (correctly predicted but uncertain) test data. On average, when facing *Type1* test data, the test relative coverage of test selection methods drops 52.99%.

C. RQ3: Impact of Type2 Test Data

Next, we explore how the data that are wrongly classified but with high prediction confidence affect the effectiveness of fault detection methods. Same as the last study, we inject the same number of labeling budget of *Type2* test data into the original test data and then perform the fault detection.

Table VII presents the results of fault detection, where All is the TRC of the entire test set, and $Type2\ only$ shows the percentage of type2 faults that have been detected. We can see that MC, DeepGini, MaxP, and PRIMA cannot detect the *Type2* faults where their $Type2\ only$ scores are nearly 0 across all the datasets and models. This means that these methods can not detect faults that are far away from the decision boundaries. Although other methods can detect some *Type2* faults, most of them detect fewer than the random selection except TestRank. From the average results, we can see that compared to the TRC values on the original test data (Ori), the TRC values on *Type2* data drop 39.80% (46.70% if without random selection). Similar to the analysis of *Type1* data, we compare the performance of all methods on *Type2* data and original test data using t-test. The results demonstrate that except for *Random* and *TestRank*, all methods perform significantly worse on *Type2* data than on original test data (with a p-value <0.05). Surprisingly, the intermediate output-based method, DSA has the greatest performance degradation. And the scores of $Type2\ only$ and All are close to the scores achieved by random selection (0.09), which indicates that when facing the *Type2* test data, DSA is completely ineffective. The potential reason could be that the distance map between the test data and the training data is significantly changed by the *Type2* data and DSA is confused by the new distance map. The deeper analysis is an interesting research direction for our future work.

Answer to RQ3: Existing fault detection methods cannot detect *Type2* faults where the model has high confidence. On average, when facing *Type2* test data, the test relative coverage scores achieved by test selection methods drop 39.80%.

D. RQ4: Model Repair

After finding the fault, the next step is to fix the fault. The most common approach the existing works [4], [13]–[15], [17] apply is to use faults as *patches* to repair (by retraining) the pre-trained model. In this part, we study how *Type1* and *Type2* data affect the effectiveness of test selection-based model retraining. Table VIII presents the detailed accuracy difference between pre-trained and re-trained models. First, it is interesting that when facing the *Type1* test data, after model retraining, most of the models have accuracy degradation (55 out of 80 models). From the average results, we can see only ATS, DeepGini, MaxP, MCP, and MC can repair the pre-trained models but with negligible improvement (only by up to 0.90% accuracy improvement). After significance analysis using t-test, we found that compared to retraining with original test data, all the methods (except for ATS) produce significantly worse models after retraining with *Type1* data. On the other hand, for the *Type2* test data, although most (70 out of 80) of the test selection methods can repair the pre-trained models and achieve positive accuracy improvement, compared to the performance of model repair on the original test data, the improvement is slight (i.e., only in 2 out of 80 cases, the results on *Type2* data are better than the results on original data). However, different from the *Type1* data, the t-test results demonstrate that only DSA produces significantly worse models after retraining on *Type2* data. This phenomenon indicates that, for model retraining, *Type1* data is more harmful than *Type2* data.

Figure 4 depicts the average accuracy improvements of model repair on all datasets and models. The results clearly show that when the test set contains *Type1* and *Type2* data, the effectiveness of test selection-based model repair is worse than when there are only original test data. Considering different test selection methods, we can see that the accuracy improvement by PRIMA and TestRank is always lower than by random selection. Note that in their original work, they did not check the effectiveness of test selection-based model repair. This reminds us that when proposing new fault detection target test selection methods, we should also explore whether the revealed faults are useful for repairing the model or not. For DSA, although it achieves better results than random selection on the original test data, its performance on the *Type1* and *Type2* test data are worse than random selection. This indicates, for model repair, that DSA is only suitable for standard test data that share the same characteristics as the training set.

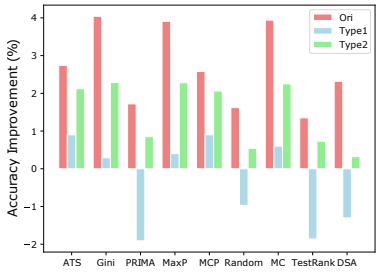


Fig. 4. Repair results by different test selection methods.

TABLE VII
RESULTS OF TRC ON *Type2* TEST DATA (LABELING BUDGET 10%).

			Random	DSA	MC	DeepGini	MCP	MaxP	TestRank	PRIMA	ATS	Average
MNSIT	**LeNet1**	**All**	0.11	0.13	0.18	0.18	0.24	0.18	0.24	0.14	0.18	0.17
		Type2 only	0.08	0.11	0.00	0.00	0.07	0.00	0.20	0.04	0.00	0.06
	LeNet5	**All**	0.10	0.10	0.10	0.11	0.20	0.11	0.16	0.12	0.15	0.13
		Type2 only	0.09	0.09	0.00	0.00	0.10	0.00	0.12	0.06	0.06	0.06
SVHN	**LeNet5**	**All**	0.22	0.21	0.50	0.57	0.53	0.58	0.46	0.45	0.57	0.45
		Type2 only	0.11	0.09	0.04	0.00	0.01	0.00	0.00	0.08	0.00	0.04
	ResNet20	**All**	0.14	0.13	0.26	0.30	0.30	0.30	0.90	0.25	0.30	0.32
		Type2 only	0.11	0.09	0.00	0.00	0.04	0.00	0.87	0.00	0.00	0.12
CIFAR10	**ResNet20**	**All**	0.13	0.12	0.52	0.50	0.40	0.49	0.23	0.23	0.53	0.35
		Type2 only	0.09	0.09	0.01	0.00	0.01	0.00	0.01	0.05	0.00	0.03
	VGG16	**All**	0.17	0.17	0.45	0.46	0.36	0.46	0.26	0.22	0.44	0.33
		Type2 only	0.09	0.10	0.00	0.00	0.01	0.00	0.07	0.07	0.00	0.04
Traffic	**LeNet5**	**All**	0.29	0.30	0.74	0.79	0.73	0.79	0.39	0.36	0.78	0.57
		Type2 only	0.09	0.09	0.00	0.00	0.09	0.00	0.00	0.08	0.00	0.04
	VGG16	**All**	0.17	0.17	0.55	0.56	0.49	0.56	0.16	0.18	0.54	0.37
		Type2 only	0.10	0.09	0.00	0.00	0.13	0.00	0.00	0.08	0.00	0.04
CIFAR100	**ResNet50**	**All**	0.37	0.34	0.70	0.79	0.70	0.79	0.31	0.48	0.74	0.58
		Type2 only	0.10	0.10	0.00	0.00	0.00	0.00	0.07	0.09	0.00	0.04
	DenseNet121	**All**	0.20	0.18	0.60	0.63	0.61	0.65	0.16	0.39	0.58	0.45
		Type2 only	0.08	0.08	0.00	0.00	0.00	0.00	0.06	0.10	0.00	0.04
Average		**Type2 only**	0.09	0.09	0.01	0.00	0.05	0.00	0.14	0.07	0.01	0.05
		Ori	0.14	0.61	0.64	0.69	0.93	0.70	0.76	0.55	0.67	0.63
		All	0.18	0.18	0.42	0.45	0.41	0.45	0.30	0.26	0.44	0.34
		Diff	22.28% ↑	71.40% ↓	35.85% ↓	55.58% ↓	35.16% ↓	35.78% ↓	59.86% ↓	52.33% ↓	34.55% ↓	39.80% ↓

TABLE VIII
ACCURACY DIFFERENCE (%) AND THE VARIANCE (IN BRACKETS) BETWEEN THE PRE-TRAINED AND RETRAINED MODELS (LABELING BUDGET 10%). THE BEST RESULTS AMONG ORI, *Type1*, AND *Type2* ARE HIGHLIGHTED USING GRAY BACKGROUND. MODELS THAT HAVE ACCURACY DEGRADATION AFTER RETRAINING ARE HIGHLIGHTED USING ORANGE BACKGROUND.

			Random	DSA	DeepGini	MCP	MC	MaxP	TestRank	PRIMA	ATS	Average
MNIST	**LeNet1**	**Ori**	0.45 (0.05)	0.59 (0.06)	0.62 (0.07)	0.67 (0.1)	0.78 (0.06)	0.74 (0.08)	0.43 (0.06)	0.55 (0.02)	0.69 (0.04)	0.61 (0.06)
		Type1	-1.60 (0.06)	-1.66 (0.37)	0.22 (0.08)	-0.29 (0.51)	0.25 (0.02)	0.25 (0.08)	-1.91 (0.28)	-2.95 (0.82)	-2.85 (0.89)	-1.17 (0.34)
		Type2	0.12 (0.2)	0.24 (0.11)	0.51 (0.12)	0.52 (0.03)	0.52 (0.09)	0.53 (0.2)	0.34 (0.11)	0.48 (0.08)	0.44 (0.16)	0.41 (0.12)
	LeNet5	**Ori**	0.17 (0.06)	0.94 (0.08)	0.87 (0.22)	0.91 (0.09)	0.85 (0.08)	0.81 (0.12)	0.42 (0.02)	0.48 (0.02)	0.72 (0.05)	0.68 (0.08)
		Type1	-1.00 (1.01)	-0.33 (0.59)	-0.03 (0.04)	-0.64 (0.41)	0.21 (0.08)	-0.03 (0.07)	-3.02 (0.55)	-1.83 (0.7)	0.33 (0.11)	-0.71 (0.40)
		Type2	0.04 (0.12)	-0.11 (0.09)	0.74 (0.13)	0.70 (0.05)	0.57 (0.20)	0.80 (0.10)	0.14 (0.16)	0.29 (0.18)	0.57 (0.30)	0.42 (0.15)
SVHN	**LeNet5**	**Ori**	0.98 (0.30)	2.19 (0.55)	4.46 (0.34)	3.81 (0.18)	3.60 (0.27)	4.27 (0.18)	3.34 (0.32)	1.15 (0.75)	2.65 (0.20)	2.94 (0.34)
		Type1	-2.29 (0.81)	-3.18 (0.49)	-2.97 (0.57)	-0.79 (2.28)	0.30 (0.89)	-0.85 (0.49)	-0.57 (0.46)	-5.02 (0.15)	-2.90 (1.38)	-2.03 (0.83)
		Type2	-0.50 (0.11)	-0.01 (0.15)	1.99 (0.32)	1.74 (0.43)	1.72 (0.28)	1.46 (0.89)	1.31 (0.32)	-0.25 (0.95)	0.67 (0.72)	0.90 (0.46)
	ResNet20	**Ori**	0.15 (0.28)	0.80 (0.20)	1.33 (0.25)	1.32 (0.31)	0.95 (0.36)	1.43 (0.12)	0.61 (0.47)	0.73 (0.30)	0.96 (0.26)	0.92 (0.28)
		Type1	-3.37 (1.38)	-3.60 (0.74)	-0.38 (0.63)	-0.38 (0.66)	-0.11 (0.21)	-0.48 (0.08)	-4.17 (0.31)	-3.47 (0.74)	-0.04 (0.80)	-1.78 (0.62)
		Type2	-0.02 (0.18)	-0.37 (0.11)	0.59 (0.21)	0.43 (0.58)	0.45 (0.52)	0.71 (0.17)	-0.16 (0.54)	0.37 (0.15)	0.56 (0.54)	0.28 (0.33)
CIFAR10	**ResNet20**	**Ori**	0.78 (0.29)	1.52 (0.35)	3.64 (0.46)	1.27 (0.26)	2.84 (0.98)	3.68 (0.62)	0.93 (0.52)	0.70 (0.36)	3.15 (0.96)	2.06 (0.53)
		Type1	-1.66 (2.04)	-3.76 (1.50)	-0.71 (0.58)	-0.63 (0.61)	-3.50 (2.44)	-1.42 (0.84)	-2.77 (2.46)	-4.54 (0.41)	-0.59 (0.10)	-2.17 (1.22)
		Type2	-0.69 (0.56)	-1.25 (1.30)	0.34 (0.35)	-0.06 (0.52)	0.98 (0.12)	0.60 (0.60)	-0.12 (0.70)	-1.09 (0.79)	0.62 (0.43)	-0.08 (0.60)
	VGG16	**Ori**	2.36 (0.17)	3.83 (0.08)	6.80 (0.19)	5.18 (0.13)	6.56 (0.17)	6.81 (0.18)	1.98 (0.11)	2.17 (0.3)	4.67 (0.11)	4.48 (0.13)
		Type1	0.75 (0.11)	0.78 (0.12)	2.09 (0.46)	1.83 (0.04)	1.40 (0.07)	1.12 (0.15)	-1.57 (0.23)	-0.50 (0.06)	3.68 (0.06)	1.06 (0.15)
		Type2	1.34 (0.04)	1.00 (0.2)	4.07 (0.04)	3.10 (0.1)	4.04 (0.03)	4.06 (0.02)	1.91 (0.10)	1.54 (0.14)	4.08 (0.07)	2.79 (0.05)
Traffic	**LeNet5**	**Ori**	6.41 (0.10)	7.25 (0.19)	10.24 (0.74)	6.79 (0.55)	10.16 (0.42)	9.85 (0.24)	2.15 (0.31)	6.97 (0.66)	7.68 (0.87)	7.50 (0.45)
		Type1	1.96 (0.60)	1.58 (1.82)	4.63 (0.71)	4.30 (0.62)	4.55 (0.86)	4.01 (1.02)	-0.58 (0.70)	2.02 (0.06)	5.68 (0.36)	3.13 (0.75)
		Type2	1.90 (0.00)	3.09 (1.01)	7.27 (0.10)	6.49 (0.74)	6.85 (0.19)	6.58 (0.07)	3.26 (0.10)	3.21 (0.23)	6.56 (0.59)	5.02 (0.34)
	VGG16	**Ori**	4.52 (0.28)	5.36 (0.65)	6.10 (0.18)	4.43 (1.10)	5.30 (0.68)	5.09 (0.49)	2.19 (0.03)	2.86 (0.56)	3.97 (0.72)	4.42 (0.52)
		Type1	1.02 (0.64)	0.59 (2.14)	3.41 (0.28)	3.40 (0.20)	3.01 (0.27)	2.61 (0.92)	0.27 (0.64)	-0.64 (0.83)	4.35 (0.96)	2.00 (0.76)
		Type2	3.83 (0.29)	0.67 (0.21)	3.99 (0.43)	3.89 (0.41)	4.24 (0.66)	3.64 (0.78)	1.26 (0.83)	2.88 (0.61)	4.31 (0.26)	3.19 (0.50)
CIFAR100	**ResNet50**	**Ori**	1.35 (1.30)	0.27 (0.86)	1.64 (0.60)	3.18 (3.26)	3.27 (2.04)	1.06 (0.77)	2.37 (0.61)	1.58 (0.71)	1.15 (1.60)	1.76 (1.31)
		Type1	-1.18 (1.65)	-1.45 (1.75)	-1.51 (2.55)	0.11 (0.05)	-0.02 (1.18)	0.59 (1.39)	-1.66 (0.97)	-0.95 (0.97)	1.09 (0.44)	-0.55 (1.22)
		Type2	0.57 (2.81)	1.19 (2.13)	1.36 (1.71)	1.72 (0.58)	0.98 (1.61)	2.12 (0.59)	0.87 (1.53)	1.25 (0.45)	1.63 (1.58)	1.30 (1.44)
	DenseNet101	**Ori**	0.68 (0.14)	2.75 (0.26)	8.76 (0.32)	0.88 (0.17)	9.01 (0.28)	9.23 (0.33)	0.43 (0.07)	1.75 (0.24)	4.53 (0.10)	4.22 (0.21)
		Type1	-3.26 (0.12)	-3.27 (0.17)	-1.60 (0.03)	2.97 (0.11)	0.49 (0.08)	-1.37 (0.16)	-4.48 (0.16)	-3.11 (0.19)	1.17 (0.31)	-1.39 (0.15)
		Type2	-0.69 (0.37)	-0.93 (0.19)	4.33 (0.28)	4.16 (0.24)	4.43 (0.21)	4.58 (0.25)	-0.75 (0.06)	0.71 (0.27)	3.88 (0.35)	2.19 (0.25)
Average		**Ori**	1.62 (0.30)	2.32 (0.33)	4.04 (0.34)	2.58 (0.62)	3.94 (0.53)	3.91 (0.31)	1.35 (0.25)	1.72 (0.36)	2.74 (0.49)	2.69 (0.25)
		Type1	-0.97 (0.55)	-1.30 (0.97)	0.29 (0.59)	0.90 (0.52)	0.60 (0.61)	0.40 (0.49)	-1.86 (0.68)	-1.91 (0.89)	0.90 (0.54)	-0.33 (0.64)
		Type2	0.54 (0.47)	0.32 (0.53)	2.29 (0.37)	2.06 (0.36)	2.25 (0.39)	2.28 (0.37)	0.73 (0.45)	0.85 (0.38)	2.12 (0.50)	1.49 (0.42)

Answer to RQ4: *Type1* and *Type2* test data harm the performance of selection-based model repair. Given these two types of data, DSA, PRIMA, and TestRank achieve worse repair results than random selection. Especially, more than half (55 out of 80) of repaired models occur accuracy degradation with *Type1* data.

E. RQ5: Performance Estimation

Finally, we explore how the choice of intermediate output affects the effectiveness of performance estimation methods. Table IX shows the frequency of test selection methods achieving the best results over different labeling budgets. The first two columns of values are the comparison between different intermediate layers in the same test selection method. The last three columns are the comparison between different methods. We can see that, first, by the same method, the average results suggest the second-last hidden layer as the best choice for both CES and PACE methods. However, when we check the results of each dataset and model, it is difficult to decide which layer should be used. For example, for CES, layer-2 is the best choice for MNIST-LeNet5, but layer-1 is the best one for SVHN-LeNet5. Similar to CES, for PACE, layer-3 achieves the best results on MNIST-LeNet5 while layer-2 is the best for SVHN-LeNet5. This means the choice of an intermediate layer highly impacts the results of these two methods and there is no clear conclusion on which layer we should choose when using different models. Then, if we compare different methods, the average results demonstrate that only CES with the outputs from layer-2 (0.44) can significantly outperform the random selection (0.38), which means if we choose an unsuitable intermediate layer, the results achieved by the well-

TABLE IX
FREQUENCY OF EACH TEST SELECTION METHOD ACHIEVING THE TOP-1 PERFORMANCE USING DIFFERENT INTERMEDIATE OUTPUTS. *layer-N* MEANS THE OUTPUT IS FROM THE LAST NTH LAYER.

			CES	PACE	CES	PACE	Random
MNSIT	LeNet1	layer-1	0.29	0.00	0.64	0.00	0.36
		layer-2	0.36	0.21	0.29	0.64	0.07
		layer-3	0.36	0.79	0.00	0.93	0.07
	LeNet5	layer-1	0.07	0.29	0.21	0.29	0.50
		layer-2	0.64	0.29	0.29	0.29	0.43
		layer-3	0.29	0.43	0.14	0.29	0.57
SVHN	LeNet5	layer-1	0.57	0.00	0.71	0.00	0.29
		layer-2	0.14	0.79	0.57	0.21	0.21
		layer-3	0.29	0.21	0.43	0.36	0.21
	ResNet20	layer-1	0.57	0.00	0.71	0.00	0.29
		layer-2	0.14	0.57	0.36	0.29	0.36
		layer-3	0.29	0.29	0.43	0.29	0.29
CIFAR10	ResNet20	layer-1	0.21	0.00	0.64	0.00	0.36
		layer-2	0.21	0.14	0.64	0.07	0.29
		layer-3	0.57	0.86	0.43	0.57	0.00
	VGG16	layer-1	0.29	0.00	0.57	0.00	0.43
		layer-2	0.50	0.93	0.71	0.00	0.29
		layer-3	0.21	0.07	0.50	0.00	0.50
Traffic	LeNet5	layer-1	0.21	1.00	0.57	0.00	0.43
		layer-2	0.50	0.00	0.50	0.00	0.50
		layer-3	0.29	0.00	0.64	0.00	0.36
	VGG16	layer-1	0.57	0.36	0.43	0.00	0.57
		layer-2	0.21	0.36	0.36	0.00	0.64
		layer-3	0.21	0.29	0.14	0.00	0.86
CIFAR100	ResNet50	layer-1	0.36	0.79	0.14	0.00	0.86
		layer-2	0.43	0.07	0.29	0.00	0.71
		layer-3	0.21	0.14	0.21	0.00	0.79
	DenseNet121	layer-1	0.14	0.00	0.21	0.00	0.79
		layer-2	0.50	0.93	0.36	0.36	0.29
		layer-3	0.36	0.07	0.50	0.21	0.29
Average		layer-1	0.33	0.24	0.49	0.03	0.49
		layer-2	0.36	0.43	0.44	0.19	0.38
		layer-3	0.31	0.31	0.34	0.26	0.39

designed methods are worse than random selection.

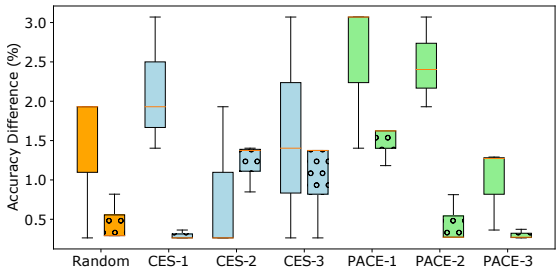


Fig. 5. Results of accuracy estimation with labeling budgets 50 and 180 (filled with circle).

In addition, we check the effectiveness of performance estimation under different labeling budgets. Figure 5 depicts the results of these three methods on labeling budgets 50 (minimum) and 180 (maximum). The results clearly show that when the labeling budget is 50, the second last hidden is more suitable for CES, but when the labeling budget is 180, the last hidden layer is the best. Besides, when selecting an unsuitable intermediate layer, the estimated results are worse than the random selection, e.g., CES-2 and PACE-1.

Answer to RQ5: Performance estimation methods are sensitive to the choice of intermediate outputs. Those methods perform worse than random selection when an unsuitable layer is chosen. Unfortunately, it is difficult to determine the best layer since there is no clear conclusion across different datasets and models.

VI. DISCUSSION

A. Guidance

Here, we provide guidelines for proposing and evaluating test selection methods:

1) From RQ2 and RQ3, we can see final output-based fault detection methods have critical constraints. To propose a fault detection method, only using the output probability is insufficient. The method can be easily fooled by uncommon data (like *Type1* and *Type2* data). It is better to avoid only relying on the output probability, e.g., combining the output and features from the input itself.
2) For learning-based methods (e.g., TestRank), the learning models (e.g., simple GNN used in TestRank) should be evaluated first to check if they can distinguish features of any type of faults (not only the faults in the original test data but also harder data like *Type1* and *Type2*) and correctly classified data.
3) RQ4 shows that when there are *Type1* data in the candidate data, retraining is ineffective and the computing resource is wasted. Thus, before repairing models via retraining, it is suggested to use out-of-distribution detection techniques [45], [46] to check the distribution of test data.
4) RQ5 demonstrates that it is difficult to choose the appropriate layer for the use of existing performance estimation methods. Therefore, when proposing performance estimation methods, a layer-selection solution should be developed along with the method. It is impractical to use all intermediate outputs due to the high complexity.

B. Threats to Validity

The **external threat** lies in the considered test selection methods, datasets, and models. For test selection methods, we collect methods that are specifically designed for test selection. Others such as neural coverage methods and active learning methods are not considered since they are proposed for different targets. For datasets and models, we use 5 datasets spanning from digit recognition to more practical traffic sign classification. And for each dataset, we include two model architectures, which can, to some extent, alleviate the model dependency. The **internal threat** can be the implementation of test selection methods and the GA-based test generation algorithm. All implementations of selection methods are modified from the official projects provided by corresponding authors. The implementation of GA-based test generation is also based on an existing work [31]. The **construct threat** can be the configuration of test selection methods and test generation process. For test selection methods, we follow their original papers and use default settings. For the configuration of the test generation process, we set parameters adaptively given the dataset and model. Note that our target is not to find a perfect parameter setting to attack test selection methods, instead, we focus on proving that non-robust features are easy to reveal.

VII. CONCLUSION

We identified and systematically assessed three types of pitfalls in existing test selection methods for reliable testing of deep learning (DL)-based systems. Via an exploratory study, we found that methods for fault detection skip faults (up to 91%) if a DL model is confident in the misclassification and introduce fake faults (up to 100%) if a model has low

confidence. In addition, selected data can degrade the accuracy when repairing models (e.g., 5.02% degradation). On the other hand, methods for performance estimation fail to defeat the simplest random selection when using an inappropriate intermediate layer. Ultimately, we provide actionable guidelines on how to mitigate pitfalls when applying existing selection methods and avoid pitfalls when developing new ones.

REFERENCES

[1] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales, "When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 142–150.

[2] W. Wu and R. Yan, "Deep chit-chat: Deep learning for chatbots," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 1413–1414.

[3] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021.

[4] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–22, 2021.

[5] V. Riccio and P. Tonella, "When and why test generators for deep learning produce invalid inputs: an empirical study," *ICSE*, 2023.

[6] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *International Conference on Learning Representations*, 2019.

[7] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *Advances in neural information processing systems*, vol. 32, 2019.

[8] M. Chen, K. Goel, N. S. Sohoni, F. Poms, K. Fatahalian, and C. Ré, "Mandoline: Model evaluation under distribution shift," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1617–1629.

[9] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. Ieee, 2017, pp. 39–57.

[10] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.

[11] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 3–18.

[12] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," *Advances in neural information processing systems*, vol. 31, 2018.

[13] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.

[14] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 177–188.

[15] W. Shen, Y. Li, L. Chen, Y. Han, Y. Zhou, and B. Xu, "Multiple-boundary clustering and prioritization to promote neural network retraining," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 410–422.

[16] Y. LI, M. LI, Q. LAI, Y. Liu, and Q. Xu, "Testrank: bringing order into unlabeled test instances for deep learning tasks," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 20 874–20 886.

[17] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 397–409.

[18] X. Gao, Y. Feng, Y. Yin, Z. Liu, Z. Chen, and B. Xu, "Adaptive test selection for deep neural networks," in *IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 73–85.

[19] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting operational dnn testing efficiency through conditioning," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 499–509.

[20] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient deep neural network testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–35, 2020.

[21] Z. Yang, J. Shi, M. H. Asyrofi, and D. Lo, "Revisiting neuron coverage metrics and quality of deep neural networks," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 408–419.

[22] S. Yan, G. Tao, X. Liu, J. Zhai, S. Ma, L. Xu, and X. Zhang, "Correlations between deep neural network model coverage criteria and model quality," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 775–787.

[23] Y. Dong, P. Zhang, J. Wang, S. Liu, J. Sun, J. Hao, X. Wang, L. Wang, J. Dong, and T. Dai, "An empirical study on correlation between coverage and robustness for deep neural networks," in *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2020, pp. 73–82.

[24] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–18. [Online]. Available: https://doi-org.proxy.bnl.lu/10.1145/3132747.3132785

[25] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[26] V. Riccio, N. Humbatova, G. Jahangirova, and P. Tonella, "Deepmetis: Augmenting a deep learning test set to increase its mutation score," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 355–367.

[27] Q. Hu, Y. Guo, M. Cordy, X. Xie, L. Ma, M. Papadakis, and Y. Le Traon, "An empirical study on data distribution-aware test selection for deep learning enhancement," *ACM Transactions on Software Engineering and Methodology*, 2022.

[28] M. Weiss and P. Tonella, "Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)," *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2022.

[29] Q. Hu, Y. Guo, M. Cordy, X. Xie, W. Ma, M. Papadakis, and Y. L. Traon, "Towards exploring the limitations of active learning: an empirical study," in *The 36th IEEE/ACM International Conference on Automated Software Engineering.*, 2021.

[30] A. Siddhant and Z. C. Lipton, "Deep Bayesian active learning for natural language processing: Results of a large-scale empirical study," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2904–2909. [Online]. Available: https://aclanthology.org/D18-1318

[31] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 739–751.

[32] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1147–1158.

[33] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning

applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Tech. Rep., 2009.

[37] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: the german traffic sign detection benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[38] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations*, 2014.

[41] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[42] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, "Diffchaser: detecting disagreements for deep neural networks." in *IJCAI*, 2019, pp. 5772–5778.

[43] D. B. Owen, "The power of student's t-test," *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 320–333, 1965.

[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[45] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Hkg4TI9xl

[46] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 7167–7177.