Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis

Zan Wang
College of Intelligence and Computing
Tianjin University
Tianjin, China
wangzan@tju.edu.cn

Hanmo You
College of Intelligence and Computing
Tianjin University
Tianjin, China
youhanmo@tju.edu.cn

Junjie Chen†
College of Intelligence and Computing
Tianjin University
Tianjin, China
junjiechen@tju.edu.cn

Yingyi Zhang
College of Intelligence and Computing
Tianjin University
Tianjin, China
yingyizhang@tju.edu.cn

Xuyuan Dong
Information and Network Center
Tianjin University
Tianjin, China
dongxuyuan@tju.edu.cn

Wenbin Zhang
Information and Network Center
Tianjin University
Tianjin, China
zhangwenbin@tju.edu.cn

***Abstract*—Deep Neural Network (DNN) testing is one of the most widely-used ways to guarantee the quality of DNNs. However, labeling test inputs to check the correctness of DNN prediction is very costly, which could largely affect the efficiency of DNN testing, even the whole process of DNN development. To relieve the labeling-cost problem, we propose a novel test input prioritization approach (called PRIMA) for DNNs via intelligent mutation analysis in order to label more bug-revealing test inputs earlier for a limited time, which facilitates to improve the efficiency of DNN testing. PRIMA is based on the key insight: a test input that is able to kill many *mutated models* and produce different prediction results with many *mutated inputs*, is more likely to reveal DNN bugs, and thus it should be prioritized higher. After obtaining a number of mutation results from a series of our designed model and input mutation rules for each test input, PRIMA further incorporates learning-to-rank (a kind of supervised machine learning to solve ranking problems) to intelligently combine these mutation results for effective test input prioritization. We conducted an extensive study based on 36 popular subjects by carefully considering their diversity from five dimensions (i.e., different domains of test inputs, different DNN tasks, different network structures, different types of test inputs, and different training scenarios). Our experimental results demonstrate the effectiveness of PRIMA, significantly outperforming the state-of-the-art approaches (with the average improvement of 8.50%∼131.01% in terms of prioritization effectiveness). In particular, we have applied PRIMA to the practical autonomous-vehicle testing in a large motor company, and the results on 4 real-world scene-recognition models in autonomous vehicles further confirm the practicability of PRIMA.**

***Index Terms*—Test Prioritization, Deep Neural Network, Mutation, Label, Deep Learning Testing**

I. Introduction

In recent years, deep neural networks (DNNs) have gained great success in many domains, e.g., autonomous vehicles [1], [2], face recognition [3], speech recognition [4], medical diagnosis [5], and software engineering [6]–[9] Unfortunately, like traditional software systems, DNNs also contain bugs [10]–[13]. Due to the popularity and importance of DNNs, DNN bugs could lead to serious consequences in practice, even disasters in safety-critical domains. For example, an Uber autonomous vehicle killed a pedestrian in Tempe, Arizona in 2018 [14]. Therefore, it is very critical to guarantee the quality of DNNs.

DNN testing is one of the most widely-used ways to guarantee the quality of DNNs [15]. In the literature, most works on DNN testing focus on proposing various metrics to measure the adequacy of test inputs [10], [11], [16] or designing various approaches to generating test inputs [13], [17]. However, beyond that, there is another key challenge in the field of DNN testing — it is very costly to label test inputs to check the correctness of DNN prediction, which could largely affect the efficiency of DNN testing, even the whole process of DNN development [18], [19]. More specifically, the reasons for the labeling-cost problem are threefold: 1) The test set is large-scale; 2) The main way of labeling is manual analysis, which tends to involve multiple persons to label one test input so as to ensure the labeling correctness; 3) Domain-specific knowledge is usually required for labeling, which makes labeling more expensive by employing professional persons. According to the existing study [18], this challenge is even more troublesome in practice, but currently few efforts have been devoted to solving it.

To relieve this problem, it is intuitive to prioritize test inputs so that those test inputs that are more likely to be incorrectly predicted by the DNN under test (also called bug-revealing test inputs) can be labeled earlier. In this way, more bug-revealing test inputs can be identified for a limited time, and in the meanwhile identifying bug-revealing test inputs earlier facilitates to conduct the debugging process earlier, which could largely improve the efficiency of DNN testing and shorten the period of DNN development. Indeed, some test input prioritization approaches for DNNs have been proposed recently to solve

†Junjie Chen is the corresponding author.

the labeling-cost problem [19]–[21], including coverage-based and confidence-based test input prioritization approaches.

However, these existing approaches either suffer from the effectiveness issue or have limited application scenarios. More specifically, coverage-based test input prioritization, which prioritizes test inputs based on their neuron coverage by adapting the coverage-based test prioritization in traditional software systems [22], [23], has been demonstrated to be not effective compared with confidence-based test input prioritization in the existing study [19]. Confidence-based test input prioritization, i.e., DeepGini, is the state-of-the-art approach for DNNs, which prioritizes test inputs for classification models by measuring the DNN's confidence about its classification for each test input [19]. That is, if a DNN model outputs more similar probabilities for all classes when classifying a test input, it means that its confidence for classifying the test input is less and thus the test input should have a higher priority for labeling. Although DeepGini has been demonstrated to be effective in some cases, it actually suffers from the issues of limited application scenarios:

- DeepGini is designed specific to classification models, and thus it may not be directly applicable to regression models, which are also widely-used in practice. In the meanwhile, it has been only evaluated on image-classification models, and thus it is unclear whether it can still perform well on other domains of test inputs, such as sequential data.
- The assumption of DeepGini that bug-revealing test inputs are predicted with similar probabilities for all classes, is not tenable in many practical cases. For example, when training data is polluted or the training scenario is transfer learning, the confidence of the DNN model for incorrect prediction tends to be strong. Moreover, many adversarial input generation methods (such as C&W [24]) aim to generate the test inputs that make the probability of the wrong class large as much as possible, which also goes against its assumption. In these practical cases, the performance of DeepGini can drop largely, which has been demonstrated in our study (Section IV-E).

Therefore, it is still desired for a more general and better test input prioritization approach for DNNs.

To further improve the performance of test input prioritization for DNNs, in this paper we propose a novel test input prioritization approach for DNNs via intelligent mutation analysis, which is called **PRIMA** (**PR**ioritizing test inputs via **I**ntelligent **M**utation **A**nalysis). The key insights of PRIMA are twofold: 1) If a test input can kill many mutated models (i.e., when the prediction result of a test input is different between the original model and the mutated model by slightly changing the original model, we regard that the test input kills the mutated model), indicating that the test input can test the model sufficiently, the test input is more likely to reveal DNN bugs. That reflects the exploration degree of the test input to the DNN model under test. 2) If many mutated test inputs by slightly changing an original test input have different prediction results with the original one on the DNN model under test, indicating that much information of the test input is utilized by the model, the test input is more sensitive to capture DNN bugs. That reflects the exploration degree to the test input itself. Based on the key insights of exploring both DNN models and test inputs, we first design a series of model mutation rules and input mutation rules in PRIMA. Based on these mutation rules, PRIMA produces a number of mutation results for each test input, and then a follow-up challenge is how to effectively utilize these mutation results to prioritize test inputs. In particular, different DNN models have different characteristics and thus their ways of utilizing mutation results to achieve the best prioritization effectiveness could be different, further indicating the difficulty of solving this challenge. In this paper, PRIMA incorporates learning-to-rank [25] (a kind of supervised machine learning to solve ranking problems) to build a ranking model, which can effectively prioritize test inputs by intelligently learning how to utilize mutation results for different DNN models. Based on the prioritization result through PRIMA, the bug-revealing test inputs can be labeled earlier so that the efficiency of DNN testing can be largely improved and the period of DNN development can be effectively shortened.

To evaluate the performance of PRIMA, we conducted an extensive study based on 36 popular subjects (we call a pair of dataset and DNN model a subject). In particular, we carefully considered the diversity of our subjects from five dimensions, including: 1) different domains of test inputs (i.e., images, text, and predefined features), 2) different types of test inputs (i.e., natural test inputs and adversarial test inputs generated by different adversarial input generation methods), 3) different tasks of DNN models (i.e., classification models and regression models), 4) different network structures of DNN models (i.e., CNN and RNN), and 5) different training scenarios (i.e., normal training, training with polluted training data, and transfer learning). To our best knowledge, this is the largest and the most diverse study in this field. Our experimental results show that PRIMA performs stably well on such diverse subjects and significantly outperforms all the compared approaches (with the average improvement of 8.50%$\sim$131.01% in terms of RAUC to be introduced in Section IV-D). For example, PRIMA performs the best among all the studied approaches in 94.44% cases. In particular, we have applied PRIMA to one of the most influential company for autonomous vehicles all over the world. Due to the company policy, we hide its name and call it $\mathcal{T}$. Our results on 4 real-world scene-recognition subjects in autonomous vehicles further confirm the practicability of PRIMA.

This work makes the following major contributions:

- **Approach**. We propose a novel test prioritization approach for DNNs via intelligent mutation analysis. In particular, we design a series of model and input mutation rules and adopt learning-to-rank to intelligently combine mutation results for effective test input prioritization.
- **Study**. We conduct the most large-scale study based on 36 subjects, in which the diversity of subjects is considered carefully from five dimensions, demonstrating the effective-

ness of our proposed approach.
- **Practical Evaluation**. We have applied our proposed approach to one of the most influential company for autonomous vehicles all over the world, further confirming its practicability.

II. Background and Related Work

A. DNN and DNN Testing

DNN consists of many layers, each of which contains a large number of neurons [26]. The neurons between layers are connected with links, each of which is equipped with a weight. These weights are learned via the training process based on training data. In general, DNN is divided into Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). DNN testing is one of the most widely-used methods to guarantee the DNN quality [10], [11], [27]–[30]. In DNN testing, test inputs refer to the inputs to be predicted by the DNN model under test, which can be different forms (e.g., images and text) according to different domains. Regarding test oracles in DNN testing, it is required for each test input to manually label its ground truth. Then, it can determine whether a test input is predicted correctly by the model by comparing the labeled ground truth and the predicted result.

B. Test Input Prioritization for DNNs

In the literature, several test input prioritization approaches for DNNs have been proposed to solve the labeling-cost problem [19]–[21]. For example, the state-of-the-art approach is DeepGini [19], which prioritizes test inputs by measuring the DNN's confidence for classifying each test input. More specifically, the test inputs that are predicted with more similar probabilities for all classes are prioritized higher. Moreover, they implemented neuron-coverage-based test input prioritization by adapting coverage-based test prioritization in traditional software for comparison in the study. Besides, Byun et al. [20] proposed to utilize more advanced metrics, i.e., surprise adequacy including LSA (Likelihood-based Surprise Adequacy) and DSA (Distance-based Surprise Adequacy) [31], to prioritize test inputs for DNNs. LSA refers to the surprise of a test input with respect to the estimated density of each activation value in a set of activation traces for training data, while DSA is defined by the Euclidean distance between the activation trace for a test input and a set of activation traces for training data. The surprise-based test input prioritization prioritizes the test inputs with larger surprise-adequacy values higher. Different from them, our work proposes PRIMA, a novel and more effective test input prioritization approach for DNNs via *mutation analysis* and *learning-to-rank*.

Furthermore, some test input selection approaches are also proposed to improve the efficiency of DNN testing [12], [18], [32]–[34], which aims to estimate the accuracy of a DNN model by selecting a small set of test inputs. Different from them, our work aims to identify more bug-revealing test inputs earlier by prioritizing test inputs.

C. Mutation-based Test Prioritization for Traditional Software

In the field of test prioritization for traditional software [22], [35]–[46], some mutation-based test prioritization approaches have been proposed [47], [48]. For example, Lou et al. [47] proposed to prioritize tests according to the number of mutation faults that are killed by each test. Shin et al. [48] also evaluated the performance of multi-objective mutation-based test prioritization, which prioritizes tests by considering both killed mutation faults and distinguished mutation faults. Our proposed approach PRIMA mainly has the following differences from them: 1) PRIMA considers to mutate both the DNN model under test and the test inputs, while traditional mutation-based test prioritization approaches mainly mutate the software under test. 2) DNN models have different characteristics from traditional software, and thus they have totally different mutation rules. 3) PRIMA incorporates learning-to-rank to intelligently utilize mutation results for test input prioritization, which is not adopted by traditional mutation-based approaches.

III. Approach

To relieve the labeling-cost problem, we propose a novel test input prioritization approach for DNNs via intelligent mutation analysis, called **PRIMA**. PRIMA considers both model mutation and input mutation based on the following two key insights: 1) If a test input can kill many mutated models (by slightly changing the model under test), indicating that the test input can test the model sufficiently, the test input is likely to reveal DNN bugs. 2) If many mutated test inputs from one test input (by slightly changing the test input) have different prediction results with the original one on the model under test, indicating that much information of the test input is effectively utilized by the model, the test input is sensitive to capture DNN bugs. They reflect the exploration degree of the test input to the DNN model under test and the test input itself respectively. Based on these key insights, PRIMA consists of three steps: 1) we design a series of model mutation rules and input mutation rules, and PRIMA obtains mutation results for each test input (Section III-A); 2) PRIMA extracts a set of features from these mutation results for each test input in order to effectively utilize these mutation results to prioritize test inputs (Section III-B); 3) PRIMA adopts the framework of learning-to-rank to build a ranking model, which is able to intelligently utilize the extracted features, for prioritizing test inputs (Section III-C). Also, we present the usage of PRIMA in Section III-D. Figure 1 shows the overview of PRIMA.

A. Mutation Rules

In PRIMA, we design two categories of mutation rules based on the above key insights, i.e., model mutation and input mutation. Model mutation is to slightly change the DNN model under test to produce a mutated model. If a test input produces different prediction results between the original model and the mutated model, we regard that the test input kills the mutated model, indicating that the slightly mutated part is effectively tested by the test input. Input mutation is to

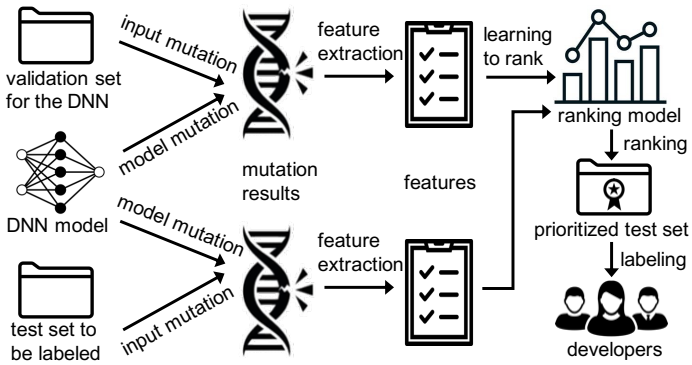

Fig. 1: Overview of PRIMA

slightly change a test input to produce a mutated test input. If the slight difference between the original test input and the mutated test input leads to different prediction results on the model under test, it means that the slightly mutated part is effectively utilized by the model and is contributing to the model. Our designed model mutation rules and input mutation rules will be introduced in detail afterwards.

1) Model Mutation Rules: As presented in Section II-A, neurons and weights are basic elements in DNNs. Thus following the existing work [49] we design four model mutation rules (which are operated at the level of neurons rather than layers and thus could conduct more fine-grained mutation as presented in the existing work [49]) on them as follows:

- *Neuron Activation Inverse (NAI)*: inverts the activation state of a neuron by changing the sign of the neuron output before passing it to the activation function.
- *Neuron Effect Block (NEB)*: blocks the effect of a neuron on the next layers by setting the neuron weights to the next layers to be 0.
- *Gauss Fuzzing (GF)*: adds noise to the weights of a neuron following Gaussian distribution $N(\mu, \delta^2)$. If δ is large, the added noise is large, which tends to produce an invalid model. Thus, we set μ to be 0 and δ to be 0.1.
- *Weights Shuffling (WS)*: shuffles the weights of a neuron with the previous layers.

In particular, PRIMA aims to slightly change the model under test since 1) largely changing the model is likely to produce invalid models; 2) slight mutation is better to simulate real bugs; 3) slight mutation is helpful to learn which parts in the model are effectively tested by each test input. Therefore, instead of changing all the neurons/weights, PRIMA randomly samples $x\%$ neurons/weights for each mutation.

2) Input Mutation Rules: The test inputs of DNNs can have different forms in different domains, such as images, sequential data (e.g., text and speech), and predefined features (like the inputs for traditional machine learning [50]). To design a general and practical approach, PRIMA should be able to handle various domains of test inputs. In particular, PRIMA considers three popular domains, including images, sequential data (i.e., we use text as the representative), and predefined features, but it is easy to extend PRIMA to more domains (to be discussed in Section VI). Due to their significant differences, we design input mutation rules for each of the three domains based on their own characteristics.

Even though there are different mutation rules for different domains of test inputs, they share the same high-level idea of designing input mutation rules — changing a small part of basic elements (such as pixels for images and characters for text) in a test input for each mutation so as to learn how much information in a test input is really contributing to the model under test. That is, each mutation rule is to select a small part of basic elements from a test input to mutate. In particular, the mutation operators for the selected basic elements are adapted from the widely-used operators used for generating adversarial inputs in the corresponding domains [51]–[55]. The reason for this idea is that if slight mutation on a test input can change the prediction results, it means that the mutated part is indeed utilized by the model and thus it could be sensitive to capture the bugs in the model. The detailed mutation rules are introduced as follows:

- *Image mutation rules*:
  - *Pixel Gauss Fuzzing (PGF)*: adds noise to the selected pixels following Guassian distribution $N(\mu, \delta^2)$. Here, we set μ to be 0 and δ to be 0.8.
  - *Pixels Shuffling (PS)*: shuffles the selected pixels.
  - *Coloring Pixel White (CPW)*: makes the colors of the selected pixels become white.
  - *Coloring Pixel Black (CPB)*: makes the colors of the selected pixels become black.
  - *Pixel Color Reverse (PCR)*: reverses the colors of the selected pixels.
- *Text mutation rules*:
  - *Character Shuffling (CS)*: shuffles the selected characters.
  - *Character Replacement (CRL)*: replaces the selected characters with other characters randomly selected from the whole set.
  - *Character Repetition (CRE)*: repeats the selected characters.
- *Predefined-features mutation rules*:
  - *Discrete Value Replacement (DVR)*: replaces the selected feature values with the other values randomly selected from the whole set of discrete values.
  - *Continuous Value Modification (CVM)*: randomly increases or decreases the selected feature values by $e\%$. To slightly mutate test inputs, we set e to be 10.

According to the designed rules, for a given test set and a model under test, PRIMA applies each model mutation rule to generate m mutated models and then obtains the prediction results of each test input on all the mutated models and the original model. Also, for each test input PRIMA applies each input mutation rule to generate n mutated inputs and then obtains the prediction results of all the mutated inputs and the original input on the model under test. That is, each test input has both model mutation results and input mutation results. In the future, more effective mutation rules [56], [57] could be

incorporated by PRIMA.

B. Feature Extraction

After obtaining the mutation results for each test input, a follow-up problem is how to effectively utilize these mutation results to prioritize test inputs. To intelligently learn how to effectively utilize these mutation results to prioritize test inputs for different models, PRIMA adopts the framework of learning-to-rank to build a ranking model for each DNN model. Since learning-to-rank requires a set of features like other supervised machine learning [25], we carefully extract a set of features from these mutation results. In this subsection, we present the step of feature extraction in PRIMA. More details about the step of learning-to-rank based ranking model building for prioritization can be found in Section III-C.

Intuitively, not only whether different prediction results (e.g., different predicted classes for classification tasks) are produced between an original model/input and a mutated model/input should be considered, but also the difference degree of the prediction results is also helpful to reflect the bug-revealing possibility. With this intuition, we identify two types of features from the mutation results for each test input. Since the outputs of classification models and regression models are different, where the former is the probabilities of a test input belonging to each class while the latter is a number, we first present the identified features for classification models as below, and then adapt them to regression models.

Given a set of test inputs $T = \{t_1, t_2, \ldots, t_s\}$, a classification model under test M with g classes (denoted as $C = \{c_1, c_2, \ldots, c_g\}$), a set of mutated models by a mutation rule (denoted as R) $M^R = \{M_1^R, M_2^R, \ldots, M_m^R\}$, a set of mutated inputs from t_k by a mutation rule (denoted as r) $t_k^r = \{t_{k1}^r, t_{k2}^r, \ldots, t_{kn}^r\}$, the predicted probabilities by t_k on M and M_j^R denoted as $P[t_kM] = \{p[t_kM]_1, p[t_kM]_2, \ldots, p[t_kM]_g\}$ and $P[t_kM_j^R] = \{p[t_kM_j^R]_1, p[t_kM_j^R]_2, \ldots, p[t_kM_j^R]_g\}$ and the corresponding predicted classes denoted as $C[t_kM]$ and $C[t_kM_j^R]$ respectively, the predicted probabilities by t_{ki}^r on M denoted as $P[t_{ki}^rM] = \{p[t_{ki}^rM]_1, p[t_{ki}^rM]_2, \ldots, p[t_{ki}^rM]_g\}$ and the corresponding predicted class denoted as $C[t_{ki}^rM]$, the features of a test input t_k for M are presented as follows:

- F_1*: Features from the mutation results in which* $C[t_kM]$ *is different from* $C[t_kM_j^R]$ *(1≤j≤m) or* $C[t_{ki}^rM]$ *(1≤i≤n) for each mutation rule*:
  - F_1^a: The number of mutants where $C[t_kM_j^R]$ (or $C[t_{ki}^rM]$) is different from $C[t_kM]$ for each model (or input) mutation rule.
  - F_1^b: The size of the set $\{C[t_kM_j^R]$ (or $C[t_{ki}^rM])$ | $C[t_kM_j^R]$ (or $C[t_{ki}^rM])$ *is different from* $C[t_kM]\}$ for each model (or input) mutation rule. This reflects the bug-revealing diversity for the test input.
  - F_1^c: The number of mutants where $C[t_kM_j^R]$ (or $C[t_{ki}^rM]$) is the class that the largest number of mutants predicts and is different from $C[t_kM]$ for each model (or input) mutation rule. This reflects the distribution of different predicted classes by these mutants to some degree.
- F_2*: Features from the difference degree between* $P[t_kM_j^R]$ *(1≤j≤m) or* $P[t_{ki}^rM]$ *(1≤i≤n) and* $P[t_kM]$ *for each mutation rule*:
  - F_2^a: The average difference degree between $P[t_kM_j^R]$ (or $P[t_{ki}^rM]$) and $P[t_kM]$, which is calculated by $\frac{\sum_{j=1}^{m} dist(P[t_kM_j^R], P[t_kM])}{m}$ (or $\frac{\sum_{i=1}^{n} dist(P[t_{ki}^rM], P[t_kM])}{n}$) for each model (or input) mutation rule, where *dist* is the cosine distance.
  - F_2^b: The distribution of all the difference degrees between $P[t_kM_j^R]$ (or $P[t_{ki}^rM]$) and $P[t_kM]$ for each model (or input) mutation rule. We split 10 equal intervals in $[0, 1]$, and then count the number of the mutants in each interval according to their difference degrees.
  - F_2^c: The average difference between the probability of $C[t_kM]$ and the probability of this class predicted by mutants for each mutation rule. The reason for this feature is that the changing probability for the predicted class by the original model and input is more relevant to the bug-revealing possibility.

Since regression models output a number rather than a class (with a list of class probabilities), we cannot extract the features like F_1, but only extract the features from the difference degree of the prediction results for each test input, including 1) the average difference of the prediction results between the original model and input and the mutants for each mutation rule, where the difference is the absolute difference between the prediction results; and 2) the distribution of the differences for each mutation rule, where we first normalize all the differences to $[0, 1]$ according to the output range of the regression model under test, then split 10 equal intervals in $[0, 1]$, and finally count the number of the mutants in each interval according to their normalized differences.

C. Learning-to-Rank based Ranking Model Building

Based on the set of features, PRIMA constructs a training set for learning-to-rank, where each instance is an input of the DNN model under test. For each instance, PRIMA extracts the above features from its corresponding mutation results, and labels it as 0 or 1 for a classification model according to whether the input is incorrectly predicted by the model under test. For a regression model, PRIMA labels each instance as the absolute difference between its prediction result and the ground-truth, where the larger difference indicates a larger bug-revealing possibility to some degree. Then, PRIMA normalizes the features to adjust values measured on different scales to a common scale. Since all the features are numeric type, PRIMA normalizes each value of these features into $[0, 1]$ using min-max normalization [58]. Suppose the set of training instances is denoted as $A = \{a_1, a_2, \ldots, a_u\}$ and the set of features is denoted as $F = \{f_1, f_2, \ldots, f_v\}$, we use x_{ij} to represent the value of the feature f_j for the instance a_i before normalization and use x_{ij}^* to represent the value of the feature f_j for the instance a_i after normalization ($1 \leq i \leq u$ and $1 \leq j \leq v$). Formula 1 presents the normalization process.

$$x_{ij}^{*} = \frac{x_{ij} - min(\{x_{kj} | 1 \leq k \leq u\})}{max(\{x_{kj} | 1 \leq k \leq u\}) - min(\{x_{kj} | 1 \leq k \leq u\})} \quad (1)$$

After obtaining the processed training set, PRIMA adopts the framework of learning-to-rank, which is a kind of supervised machine learning and has been widely used to solve ranking problems in many domains such as document retrieval [59] and expert search [60], to build a ranking model for prioritizing test inputs. In this way, the mutation results can be intelligently utilized to achieve the best prioritization effectiveness for different models. In particular, PRIMA adopts the XGBoost ranking algorithm [61], which is an optimized distributed gradient boosting learning algorithm, to build the learning-to-rank based ranking model. The reasons using this learning-to-rank algorithm are fourfold: 1) The labels in our training set for classification models are 0 or 1, which is actually the labels for classification. In particular, the XGBoost ranking algorithm is good at handling the classification labels for ranking tasks by effectively searching for optimal splits [61]. 2) It is able to effectively learn more complex features from basic features using tree ensemble models, which matches our problem well. 3) It has been demonstrated to be effective and efficient compared with other popular learning-to-rank algorithms [61]. 4) It makes the ranking results interpretable by measuring the contribution of each feature to the ranking model.

D. Usage of PRIMA

Based on the ranking model, PRIMA predicts a score for each test input in a test set and then prioritizes all these test inputs according to the descending order of their scores. In particular, before prediction through the ranking model, it is also required to extract features for each test input in the test set from its corresponding mutation results.

During the practical usage of PRIMA, we use the validation set for the DNN model under test as the training set for building a ranking model through learning-to-rank, since the ground truth whether each input in the validation set is predicted correctly is known and the validation set is independent with both training and test sets for the DNN model. After building a ranking model for the DNN model under test based on its validation set, PRIMA can use this ranking model to prioritize various test sets for the DNN model. That is, for a DNN model under test, the ranking model is built once based on its validation set and then can be constantly used for prioritizing its different sets of test inputs, whose stable effectiveness has been demonstrated in our study (Section IV-E).

IV. Experimental Study Design

In the study, we address two research questions. **RQ1** is to investigate the effectiveness of PRIMA compared with the existing approaches. Also, we analyzed the contributions of our extracted features to the overall effectiveness of PRIMA in order to further interpret its effectiveness. **RQ2** is to investigate the efficiency of PRIMA. Also, we explored how to further improve its efficiency.

A. Subjects

In our study, we used 36 pairs of datasets and DNN models as subjects. Table I presents their basic information. In particular, to sufficiently evaluate the effectiveness of PRIMA, we carefully considered the diversity of subjects from five dimensions. To our best knowledge, this is the most large-scale and diverse study in the field.

(1) Different domains of test inputs. We considered three domains of test inputs, including images, text, and predefined features. More specifically, we collected *8 image datasets*, i.e., CIFAR-10 (a 10-class ubiquitous object dataset [62]), CIFAR-100 (a 100-class ubiquitous object dataset [62]), MNIST (a handwritten digit dataset [63]), MNIST_VS_USPS (a handwritten digit dataset for transfer learning [64]), COIL (a 20-class object recognition dataset for transfer learning [65]), PIE27_VS_PIE5 (a face dataset for transfer learning [66]), PIE27_VS_PIE9 (a face dataset for transfer learning [66]), and Driving (an autonomous driving dataset provided by Udacity [67]), *5 text datasets*, i.e., TREC (a question classification dataset [68]), IMDB (a large movie review dataset for binary sentiment classification [69]), SMS Spam (a mobile phone spam messages dataset [70]), CoLA (a linguistic acceptability dataset [71]), and Hate Speech (a hate speech and offensive language collection dataset [72]), and *one dataset with predefined features*, i.e., KDDCUP99 (a network intrusion information dataset provided by a competition in KDD'99 [73]).

(2) Different tasks of DNN models. We considered both classification models (ID: 1∼24, 31∼36) and regression models (ID: 25∼30). The number of classes ranges from 2 to 100 across all the classification models.

(3) Different network structures of DNN models. We considered both CNN (ID: 1∼30, 36) and RNN (ID: 31∼35). The number of layers ranges from 5 to 100 and the number of weights ranges from 16K to 20,081K across all the models.

(4) Different types of test inputs. We considered both natural test inputs and adversarial test inputs. Here, we adopted three widely-used adversarial input generation methods, i.e., C&W (Carlini&Wagner) [24], BIM (Basic Iterative Methods) [74], and JSMA (Jacobian-based Saliency Map Attack) [75]. Besides the original test set (i.e., the set of natural test inputs), following the existing work [18], [19] we also constructed mixed test sets by mixing natural test inputs and adversarial test inputs. More specifically, for each adversarial input generation method, we first generated the same number of adversarial test inputs as the corresponding natural test inputs for CIFAR-10 and CIFAR-100, respectively. Then, we randomly selected half of natural test inputs and half of adversarial test inputs to construct a mixed test set for each of the two datasets under each adversarial input generation method. That is, we had both original test sets and mixed test sets in our study. In this way, we had four different test sets for each of them (i.e., VGG-16 based on CIFAR-10, ResNet-20 based on CIFAR-10, VGG-19 based on CIFAR-100, ResNet-32 based on CIFAR-100) respectively, which can be used to investigate whether PRIMA performs well for different test sets when building the

TABLE I: Basic information of subjects

ID	Dataset	Model	#Test	Type	Domain
1	CIFAR-10	VGG-16	10,000	original	image
2	CIFAR-10	VGG-16	10,000	+BIM	image
3	CIFAR-10	VGG-16	10,000	+C&W	image
4	CIFAR-10	VGG-16	10,000	+JSMA	image
5	CIFAR-10	ResNet-20	10,000	original	image
6	CIFAR-10	ResNet-20	10,000	+BIM	image
7	CIFAR-10	ResNet-20	10,000	+C&W	image
8	CIFAR-10	ResNet-20	10,000	+JSMA	image
9	CIFAR-100	VGG-19	10,000	original	image
10	CIFAR-100	VGG-19	10,000	+BIM	image
11	CIFAR-100	VGG-19	10,000	+C&W	image
12	CIFAR-100	VGG-19	10,000	+JSMA	image
13	CIFAR-100	ResNet-32	10,000	original	image
14	CIFAR-100	ResNet-32	10,000	+BIM	image
15	CIFAR-100	ResNet-32	10,000	+C&W	image
16	CIFAR-100	ResNet-32	10,000	+JSMA	image
17	MNIST	LeNet-5	10,000	original	image
18	MNIST-M1	LeNet-5	10,000	original	image
19	MNIST-M2	LeNet-5	10,000	original	image
20	MNIST-M3	LeNet-5	10,000	original	image
21	MNIST_VS_USPS	LeNet-5	1,800	original	image
22	COIL	VGG-11	1,000	original	image
23	PIE27_VS_PIE5	VGG-11	3,332	original	image
24	PIE27_VS_PIE9	VGG-11	1,632	original	image
25	Driving	Dave-orig	5,614	original	image
26	Driving	Dave-drop	5,614	original	image
27	Driving	Dave-orig	5,614	light	image
28	Driving	Dave-drop	5,614	light	image
29	Driving	Dave-orig	5,614	patch	image
30	Driving	Dave-drop	5,614	patch	image
31	TREC	Bi-LSTM	952	original	text
32	IMDB	Bi-LSTM	15,000	original	text
33	SMS Spam	Bi-LSTM	3,000	original	text
34	CoLA	Bi-LSTM	4,000	original	text
35	Hate Speech	Bi-LSTM	14,652	original	text
36	KDDCUP99	CNN	311,027	original	features

ranking model once (as presented in Section III-D). Besides, regarding Dave-orig and Dave-drop, we also have three test sets respectively, i.e., the original one, the patched test sets (blocking some parts of each test input), and the lighted test sets (changing the intensities of lights for each test input).

(5) Different training scenarios. We considered three training scenarios. The first one is the normal training scenario for a dataset and a DNN model. The second one is the training scenario with polluted training sets that are mutated by an accident or malicious attack (ID: 18∼20). In particular, we simulated this training scenario by adopting three polluted training sets from MNIST, which are widely-used by the existing work [12], [18]. More specifically, the three polluted training sets are produced by changing the labels of training inputs, i.e., 8⇔0, 7⇔1, and 9⇔3, respectively. The third one is the transfer-learning scenario (ID: 21∼24), which trains a model based on a training set for solving a problem and then applies this model to a different but related problem.

B. Compared Approaches

In our study, we considered three compared approaches, i.e., the state-of-the-art approach **DeepGini** and two surprise-based test input prioritization approaches (LSA-based and DSA-based approaches, denoted as **LSA** and **DSA** in this paper respectively). More details about these compared approaches can be found in Section II-B. In the existing study [19], neuron-coverage-based test input prioritization has been compared with DeepGini, where the latter outperforms the former. Therefore, we did not compare with the former in our study. However, more advanced metrics, i.e., surprise adequacy including LSA and DSA, have been used for prioritizing DNN test inputs but the surprise-based test input prioritization approaches have not been studied together with DeepGini. Therefore, we also used the surprise-based test input prioritization for comparison.

Please note that DeepGini and DSA could not directly apply to regression models. Therefore, we applied all the compared approaches to classification models and applied only LSA to regression models for comparison. Since LSA crashes when running on subjects 34∼36 due to lacking data points of certain classes (which has been confirmed by the authors of LSA), we do not include the results of LSA on these subjects.

C. Implementation and Configuration

We implemented PRIMA in Python based on Keras 2.3.1 [76] and XGBoost 1.1.1 [61], and adopted the existing implementations of all the compared approaches, which are released by the corresponding work [19], [31]. Regarding our mutation rules, we set the number of model mutants m to be 100 and the percentage of neurons/weights selected x to be 10 for model mutation. For input mutation, we set the number of selected basic elements to be 1 and the number of input mutants n to be 50 in the domains of text and pre-defined features, while we set the number of selected pixels to be 0.5% of the total number of pixels and n to be 200 in the domains of images. Here, we selected a relatively small part of basic elements to mutate in order to achieve slight mutation. We set them by conducting a preliminary study based on a small dataset, and found that such settings are effective in general. Also, for the XGBoost ranking algorithm in PRIMA, we set *learning_rate* to be 0.05, *colsample_bytree* to be 0.5, and *max_depth* to be 5. In particular, the XGBoost ranking algorithm is robust to parameter selection [61], [77]. We also investigated the influence of main parameters on PRIMA, which will be presented in Section VI-B.

As presented in Section III-D, for each subject, PRIMA uses the validation set to build a ranking model and then prioritizes test sets using the built ranking model. For the subjects whose validation sets are not available, we simulated a validation set by randomly selecting a set of inputs from the original test set and then prioritized the remaining inputs in the test set for each of these subjects.

Our experiments are conducted on the Intel Xeon Silver-4214 machine with 128GB RAM, Ubuntu 18.04, and 8 RTX 1080 Ti GPUs in parallel. Our code and experimental data can be found on our project homepage: https://github.com/sail-repos/PRIMA.

D. Measurements

To measure the prioritization effectiveness, following the existing work [20] we transformed the prioritization result produced by a test input prioritization approach for a subject

to a figure. For classification models, the x-axis of the figure is the number of prioritized test inputs and the y-axis is the number of bug-revealing test inputs, while for regression models the x-axis of the figure is also the number of prioritized test inputs but the y-axis is the accumulated difference between the predicted results and the ground-truth. Then, we calculated the ratio of the area under curve for the test input prioritization approach to the area under the curve of the ideal prioritization as the metric. We call this metric **RAUC**. Larger is better.

In particular, since labeling test inputs is very costly and the resources are limited, it is better if more bug-revealing test inputs can be identified when labeling fewer test inputs. Therefore, we used **RAUC-*n***, which refers to the RAUC for the first n prioritized test inputs, as the measurement to measure the prioritization effectiveness in our study. This measurement reflects the prioritization effectiveness under the given number of test inputs to be labelled. In our study, we considered n to be 100, 200, 300, and 500 respectively since the resources tend to be limited and thus the given number of test inputs to be labelled tends to be small. Here, we denote their RAUC-n as **RAUC-*100***, **RAUC-*200***, **RAUC-*300***, and **RAUC-*500***, respectively. We also presented the prioritization effectiveness on all the test inputs for each subject, denoted as **RAUC-all**. In particular, the results when n is set to be larger (e.g., 1000) can be found on our project homepage. Also, we will explain the reason why not use the widely-used APFD [78] in traditional test prioritization as the measurement in our study in Section VI-C.

Also, we measured the prioritization efficiency for each test input prioritization approach. In our study, we used the time spent on test input prioritization as the metric.

E. Results and Analysis

1) RQ1: Effectiveness of PRIMA: We first studied the prioritization effectiveness of PRIMA.

•**Overall effectiveness**. Since the number of subjects used in our study is large and the space is limited, we put the detailed comparison results for each subject on our project homepage. We present the overall comparison results across all the subjects in Table II. Since DeepGini and DSA could not directly apply to regression models, we separately present the overall comparison results on classification models and regression models. Among 180 (36 subjects * 5 metrics) cases, PRIMA performs the best in 94.44% (170 out of 180) cases, where Deepgini performs the best in 5.56% (10 out of 180) cases and LSA and DSA cannot perform the best in any case. In terms of all the used metrics, the average results of PRIMA on classification models range from 0.868 to 0.919 with the average improvements of 8.50%∼18.24% compared with DeepGini, 34.16%∼57.17% compared with LSA, and 27.29%∼40.23%, respectively. On regression models, the average results of PRIMA range from 0.779 to 0.808 with the average improvements of 17.27%∼131.01% compared with LSA. In particular, we conducted statistical analysis to investigate whether PRIMA significantly outperforms all the compared approaches by conducting the Wilcoxon Signed-Rank Test [79] in terms of each metric at the significance level 0.05. We found that all the p values are smaller than 0.05, indicating that PRIMA significantly outperforms all the compared approaches in terms of all the metrics in statistics. The results demonstrate the effectiveness of PRIMA.

•**Effectiveness on different domains of test inputs**. Table III shows the effectiveness of PRIMA on different domains of test inputs (i.e., images, text, and predefined features). We found that, PRIMA outperforms all the compared approaches on all the three domains in terms of various metrics. In particular, this is also the first time to study these compared approaches in the domains of text and predefined features. We found that PRIMA largely outperforms DeepGini, LSA, and DSA with the average improvements of 39.17%, 321.68%, and 47.99% in terms of RAUC-100 in the domain of text. Also, the average RAUC-All value of PRIMA achieves 0.966 while those of DeepGini and DSA are only 0.743 and 0.730 respectively in the domain of predefined features. The results demonstrate the stable effectiveness of PRIMA in various domains.

•**Effectiveness on polluted training and transfer learning**. As presented in Section I, the assumption of DeepGini can be violated in many practical scenarios, causing its effectiveness drops largely. In this paper, we are the first to study these approaches in two practical scenarios, i.e., polluted training and transfer learning scenarios, whose results are shown in Table IV. We found that PRIMA largely outperforms all the compared approaches in terms of all the metrics in both training scenarios. In the polluted training scenario, the average results of PRIMA in terms of all these metrics range from 0.983 to 1 while those of DeepGini only range from 0.443 to 0.583. In the transfer learning scenario, the average results of PRIMA in terms of all these metrics range from 0.859 to 0.919 while those of DeepGini only range from 0.740 to 0.849. The results confirm our claims in Section I, demonstrating that the assumption of DeepGini is not tenable in the two practical scenarios, especially the polluted training scenario. Moreover, the results further demonstrate the stable effectiveness of PRIMA in various practical scenarios.

•**Effectiveness on different types of test inputs**. We further investigated the effectiveness of PRIMA on different types of test inputs (i.e., natural test inputs and adversarial test inputs), which can also answer whether once one ranking model through PRIMA is built for a DNN model, it can perform well for various test sets of the DNN model. Due to the space limit, we used CIFAR-100 and VGG-19 (ID: 9∼12) as the representative and the conclusion holds for other subjects. From Table V, PRIMA performs the best among all the approaches for the four test sets in terms of various metrics. In particular, the RAUC-100 values of PRIMA achieve 0.977, 0.988, 0.972, and 0.981, respectively, demonstrating the stable effectiveness of PRIMA on different types of test inputs. That also demonstrates, for a DNN model under test, the built ranking model through PRIMA is not necessary to be retrained frequently since it can perform rather stably on different test sets, which indicates the practicability of PRIMA.

•**Feature contribution**. The ranking model in PRIMA is built

TABLE II: Overall comparison results across all the subjects

	Approach	#Best cases in RAUC-					Average RAUC-					Improvement of PRIMA (%) in RAUC-				
		100	200	300	500	All	100	200	300	500	All	100	200	300	500	All
C	DeepGini	2	2	2	1	3	0.751	0.753	0.752	0.755	0.847	18.24	16.47	15.69	14.97	8.50
	LSA	0	0	0	0	0	0.568	0.558	0.559	0.571	0.685	56.34	57.17	55.64	52.01	34.16
	DSA	0	0	0	0	0	0.648	0.632	0.625	0.619	0.722	37.04	38.77	39.20	40.23	27.29
	PRIMA	28	28	28	29	27	0.888	0.877	0.870	0.868	0.919	-	-	-	-	-
R	LSA	0	0	0	0	0	0.345	0.357	0.368	0.394	0.689	131.01	122.97	114.67	97.72	17.27
	PRIMA	6	6	6	6	6	0.797	0.796	0.790	0.779	0.808	-	-	-	-	-

* Rows "C" and "R" present the overall results on classification and regression models, respectively. Columns 3-7 present the number of subjects where each approach performs the best in terms of each metric, Columns 8-12 present the average results across all the subjects in terms of each metric, and Column 13-27 present the average improvement of PRIMA over each compared approach in terms of each metric.

TABLE III: Comparison on different domains of test inputs

Domain	Approach	Average RAUC-				
		100	200	300	500	All
Image	DeepGini	0.796	0.798	0.796	0.796	0.867
	LSA	0.564	0.551	0.549	0.555	0.688
	DSA	0.688	0.669	0.659	0.651	0.730
	PRIMA	0.898	0.892	0.888	0.886	0.916
Text	DeepGini	0.503	0.502	0.507	0.529	0.771
	LSA	0.166	0.225	0.281	0.376	0.670
	DSA	0.473	0.490	0.509	0.529	0.715
	PRIMA	0.700	0.668	0.647	0.635	0.799
Features	DeepGini	0.899	0.920	0.920	0.922	0.743
	LSA	-	-	-	-	-
	DSA	0.688	0.669	0.659	0.651	0.730
	PRIMA	1.000	0.999	0.976	0.956	0.966

TABLE IV: Comparison on polluted and transfer learning

Scenario	Approach	Average RAUC-				
		100	200	300	500	All
Polluted	DeepGini	0.561	0.512	0.484	0.443	0.583
	LSA	0.491	0.444	0.405	0.351	0.489
	DSA	0.607	0.528	0.485	0.432	0.551
	PRIMA	1.000	1.000	1.000	0.999	0.983
Transfer	DeepGini	0.760	0.753	0.740	0.754	0.849
	LSA	0.292	0.322	0.356	0.431	0.593
	DSA	0.537	0.526	0.521	0.540	0.687
	PRIMA	0.924	0.896	0.867	0.859	0.919

via the XGBoost ranking algorithm, which is interpretable by providing the contribution of each feature to the ranking model. Due to the limited space, we show Top-10 features with the largest contributions on average across all the image-classification subjects in Table VI. We found that Top-10 features contain both categories of features (presented in Section III-B) and contain both model and input mutation rules, demonstrating the rationality of PRIMA in design. Moreover, for the image-classification subjects, model mutation seems to make more contributions than input mutation, and NEB and NAI contribute more than the other model mutation rules while PS and PGF contribute more than the other input mutation rules, which can guide us to optimize PRIMA in order to further improve its effectiveness and efficiency.

TABLE V: Comparison on different types of test inputs

ID	Approach	RAUC-				
		100	200	300	500	All
9	DeepGini	0.870	0.884	0.885	0.889	0.888
	LSA	0.643	0.590	0.567	0.542	0.623
	DSA	0.715	0.720	0.707	0.684	0.737
	PRIMA	0.977	0.960	0.947	0.931	0.901
10	DeepGini	0.874	0.835	0.821	0.804	0.778
	LSA	0.913	0.909	0.909	0.902	0.848
	DSA	0.918	0.945	0.947	0.949	0.949
	PRIMA	0.988	0.984	0.983	0.983	0.953
11	DeepGini	0.905	0.927	0.933	0.942	0.897
	LSA	0.888	0.857	0.841	0.824	0.779
	DSA	0.884	0.879	0.880	0.878	0.847
	PRIMA	0.972	0.974	0.975	0.973	0.909
12	DeepGini	0.930	0.950	0.955	0.957	0.943
	LSA	0.957	0.928	0.919	0.905	0.808
	DSA	0.959	0.942	0.935	0.924	0.865
	PRIMA	0.981	0.979	0.980	0.979	0.945

TABLE VI: Top-10 features in terms of the average contribution across all the image-classification subjects

Rank	Rule	Feature	Score	Rank	Rule	Feature	Score
1	NEB	F_2^c	0.051	2	NEB	F_1^a	0.044
3	NAI	$F_2^b(0, 0.1)$	0.035	4	GF	F_1^a	0.032
5	PS	F_2^c	0.031	6	NEB	$F_2^b(0, 0.1)$	0.029
7	NEB	F_2^a	0.025	8	PGF	$F_2^b(0.2, 0.3)$	0.025
9	NAI	F_1^a	0.022	10	PGF	F_2^c	0.021

* $F_2^b(0, 0.1)$ and $F_2^b(0.2, 0.3)$ refer to the feature F_2^b in intervals $[0, 0.1)$ and $[0.2, 0.3)$ respectively.

To sum up, PRIMA outperforms all the compared approaches in general, on different tasks of DNN models, different domains of test inputs, different training scenarios, and different types of test inputs.

2) RQ2: Efficiency of PRIMA: We investigated the efficiency of PRIMA in Table VII. As DeepGini and DSA cannot apply to regression models, we compared these approaches in terms of efficiency on all the classification subjects (subjects 34∼36 are not included for LSA as explained in Section IV-B). The average time spent on test prioritization of PRIMA is 10.1 minutes while that of DeepGini is only 0.1 minutes. Although PRIMA is less efficient than DeepGini, the cost of PRIMA is

TABLE VII: Efficiency comparison across all the classification subjects (in minutes)

Approach	Mean	Std.	Min.	Max.
DeepGini	0.1	0.1	< 0.1	1.4
LSA	1.5	1.0	< 0.1	2.8
DSA	23.5	110.1	< 0.1	616.2
PRIMA	10.4	6.2	2.4	27.7

still acceptable in practice compared with the time-consuming and expensive manual labeling, which has been confirmed by our industry partners in Section V.

Actually, there is a promising direction to further improve the efficiency of PRIMA. As shown in Table VI, some mutation rules (e.g., NEB and NAI) tend to make more contributions than others to the effectiveness of PRIMA in general, indicating their different influence on PRIMA to some degree. Thus, if we only keep the mutation rules making more contributions in PRIMA, the effectiveness of PRIMA could be affected slightly while its efficiency could be improved largely. To explore the feasibility of this direction, we conducted a preliminary study by taking subject 10 as an example. The results show that, with only these top mutation rules (listed in Table VI), the efficiency of PRIMA is improved by 38.81% while its effectiveness only decreases less than 0.02 in terms of RAUC-100, demonstrating this direction is indeed promising.

To sum up, PRIMA is less efficient than DeepGini but its cost is still acceptable. Moreover, selecting and using the mutation rules making more contributions could improve the efficiency of PRIMA without much effectiveness loss.

V. Practical Evaluation

PRIMA has been applied to the practical autonomous-vehicle testing in a large motor company, which is one of the most influential company for autonomous vehicles all over the world. Due to the company policy, we hide the company name and call it $\mathcal{T}$ in this paper. This company has a large number of DNN models built by themselves for autonomous vehicles, as well as a large number of test inputs to test these models. In particular, all these test inputs are required to be labeled manually, which is very time-consuming and expensive. This company even has established a specialized department to finish the labeling task. PRIMA aims to solve the labeling-cost problem and improve the efficiency of DNN testing, which admirably serves their needs.

We have evaluated the effectiveness of PRIMA based on three real-world DNN models used for traffic scene recognition in autonomous vehicles in $\mathcal{T}$. For ease of presentation, we call the three DNN models *M1*, *M2*, and *M3* respectively. M1 is a binary classification model (detecting lane-changing behaviours of ego vehicles) with one test set (denoted as $M1_t$), whose size is larger than 50K. M2 is a classification model with four classes (recognizing different lane-changing scenarios) and it has one test set (denoted as $M2_t$), whose size is nearly 6K. M3 is a classification model with eight classes (recognizing typical behaviours of ego vehicles) and it has two test sets (denoted as $M3_{t1}$ and $M3_{t2}$), whose sizes are larger than 10K and 20K respectively. All of them use the RNN structure and the form of their test inputs is predefined features that are collected from sensors and automatically processed. Due to the company policy, we have to hide more details about these models and datasets.

TABLE VIII: Effectiveness on industrial subjects

ID	Approach	RAUC-100	RAUC-200	RAUC-300	RAUC-500	RAUC-All
$M1_t$	DeepGini	0.512	0.535	0.541	0.556	0.688
	PRIMA	0.740	0.651	0.605	0.555	0.746
$M2_t$	DeepGini	0.491	0.596	0.621	0.659	0.765
	PRIMA	0.889	0.830	0.776	0.720	0.766
$M3_{t1}$	DeepGini	0.847	0.903	0.917	0.827	0.656
	PRIMA	1.000	1.000	0.987	0.857	0.651
$M3_{t2}$	DeepGini	0.983	0.981	0.980	0.976	0.941
	PRIMA	1.000	0.995	0.990	0.986	0.935

Table VIII shows the effectiveness of PRIMA on the four industrial subjects. Here, we compared PRIMA with the state-of-the-art approach DeepGini (which has been demonstrated to be much better than both LSA and DSA on open-source subjects). We found that PRIMA also outperforms DeepGini in terms of RAUC-100/200/300/500 on all the four industrial subjects. For example, in terms of RAUC-100, PRIMA achieves 0.740, 0.889, 1, and 1 on the four subjects with the improvements of 44.53%, 81.06%, 18.06%, and 1.73% respectively. The results further confirm the effectiveness of PRIMA in practice. According to the practical evaluation, PRIMA has been largely appreciated by the developers in $\mathcal{T}$, and in particular they think that the automatic prioritization cost of PRIMA is totally acceptable compared with the time-consuming and expensive manual labeling, further confirming the practicability of PRIMA.

VI. Discussion

A. Generality of PRIMA

Our study has demonstrated the effectiveness of PRIMA based on a large number of subjects with great diversity, indicating the generality of PRIMA to some degree. Besides, although we considered three domains (i.e., images, text, and predefined features) for PRIMA in our work, PRIMA actually can be applicable to more domains (e.g., speech) as long as input mutation rules for the corresponding domains are designed. As presented in Section II-C, designing input mutation rules for different domains shares the same high-level idea, i.e., *changing a small part of basic elements* in a test input for each mutation by adapting the operators widely used for generating adversarial inputs in the corresponding domains. With this high-level idea, it is easy to extend PRIMA to more domains since it is easy to determine the basic elements of test inputs and find the widely-used operators for generating adversarial inputs in the corresponding domains. For example, for the speech domain, the basic element of a test input is

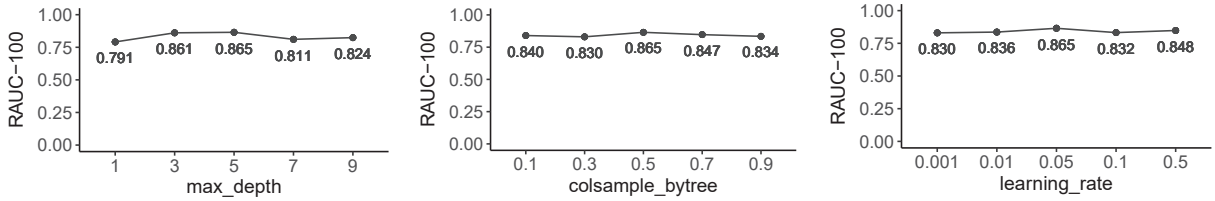

Fig. 2: Impact of main parameters in PRIMA

phoneme and the widely-used adversarial operators include adding noise to audio signal [80] and attacking phonetically similar phrases [81], and thus PRIMA could be easily extended to the speech domain. That further reflects the generality of PRIMA to some degree.

B. Impact of Main Parameters in PRIMA

We investigated the impact of main parameters in PRIMA, including *max_depth* (the maximum tree depth for each XGBoost model), *colsample_bytree* (the sampling ratio of columns of features when constructing each tree) and *learning_rate* (the boosting learning rate) in the XGBoost ranking algorithm. Here, we randomly selected six subjects (ID: 1, 4, 9, 18, 24, and 32) for this experiment by considering different domains of test inputs, different training scenarios, and different types of test inputs. Figure 2 shows the effectiveness of PRIMA under different parameter settings in terms of average RAUC-100 across the six subjects. We found that, PRIMA performs stably under different parameter settings and our default settings are also indeed proper.

C. Threats to Validity

The *internal* threat to validity mainly lies in the implementation of our approach PRIMA, all the compared approaches, and experimental scripts. To reduce this threat, we adopted the implementations of the compared approaches released by the authors, implemented PRIMA based on popular libraries (presented in Section IV-C), and carefully checked the code of PRIMA and experimental scripts.

The *external* threats to validity mainly lie in the subjects used in our study. To reduce this threat, we collected a large number of subjects with great diversity.

The *construct* threats to validity mainly lie in the parameters in PRIMA and the measurements used in our study. To reduce the threat from the parameters in PRIMA, we presented the parameter settings in Section IV-C and investigated the impact of main parameters in Section VI-B. Regarding the measurements used in our study, we measured both prioritization effectiveness and efficiency. Here, we used RAUC as the metric for effectiveness following the existing work [20] and the prioritization time as the metric for efficiency. In our study, we did not use the widely-used metric in traditional test prioritization, i.e., APFD [78] to measure the prioritization effectiveness. This is because when the accuracy of a model on a test set is low (i.e., there are a large number of bug-revealing test inputs), the upper bound of APFD could be much smaller than 1, which may affect the comparison among the subjects with very different accuracy.

VII. Conclusion

To solve the labeling-cost problem in DNN testing, we propose a novel test input prioritization approach, called PRIMA, to prioritize bug-revealing test inputs higher. PRIMA is based on mutation analysis and learning-to-rank. Its insight is that a test input that can kill many mutated models and produce different prediction results with many mutated inputs, is more likely to reveal DNN bugs, and thus it should have a higher priority for labeling. After obtaining a number of mutation results via our designed model and input rules, PRIMA incorporates learning-to-rank to build a ranking model by intelligently combining these mutation results for test input prioritization. Our results on 36 diverse subjects demonstrate the effectiveness of PRIMA and it significantly outperforms the state-of-the-art approaches. Moreover, PRIMA has been applied to practical autonomous-vehicle testing in a large motor company and the results on 4 industrial subjects further demonstrate its effectiveness in practice.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China 62002256 and 61872263, and Intelligent Manufacturing Special Fund of Tianjin 20193155, and the Fund No. 2020-JCJQ-JJ-490.

References

[1] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2015, pp. 2722–2730.

[2] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 303–314.

[3] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," in *Advances in neural information processing systems*, 2014, pp. 1988–1996.

[4] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[5] Z. Obermeyer and E. J. Emanuel, "Predicting the future—big data, machine learning, and clinical medicine," *The New England journal of medicine*, vol. 375, no. 13, p. 1216, 2016.

[6] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2019, pp. 364–375.

[7] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 2019, pp. 111–120.

[8] J. Chen, S. Zhang, X. He, Q. Lin, H. Zhang, D. Hao, Y. Kang, F. Gao, Z. Xu, Y. Dang *et al.*, "How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2020, pp. 373–384.

[9] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *The 43rd International Conference on Software Engineering*. IEEE, 2021, to appear.

[10] 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*. ACM, 2018, pp. 120–131.

[11] 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*. ACM, 2017, pp. 1–18.

[12] 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*. ACM, 2019, pp. 499–509.

[13] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 109–119.

[14] "News," Accessed: 2020. [Online]. Available: https://www.vice.com/en_us/article/9kga85/uber-is-giving-up-on-self-driving-cars-in-california-after-deadly-crash

[15] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *CoRR*, vol. abs/1906.10742, 2019.

[16] J. Chen, M. Yan, Z. Wang, Y. Kang, and Z. Wu, "Deep neural network test coverage: How far are we?" *arXiv preprint arXiv:2010.04946*, 2020.

[17] 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*. ACM, 2019, pp. 146–157.

[18] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient dnn testing," *ACM Transactions on Software Engineering and Methodology*, 2020, to appear.

[19] 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 *29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2020, pp. 177–188.

[20] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. D. Cofer, "Input prioritization for testing neural networks," in *IEEE International Conference On Artificial Intelligence Testing*. IEEE, 2019, pp. 63–70.

[21] L. Zhang, X. Sun, Y. Li, and Z. Zhang, "A noise-sensitivity-analysis-based test prioritization technique for deep neural networks," *CoRR*, vol. abs/1901.00054, 2019.

[22] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Softw. Test. Verification Reliab.*, vol. 22, no. 2, pp. 67–120, 2012.

[23] Y. Lou, J. Chen, L. Zhang, and D. Hao, "Chapter one - A survey on regression test-case prioritization," *Adv. Comput.*, vol. 113, pp. 1–46, 2019.

[24] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy*, 2017, pp. 39–57.

[25] T.-Y. Liu, *Learning to rank for information retrieval*. Springer Science & Business Media, 2011.

[26] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[27] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, 2011.

[28] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: model-based quantitative analysis of stateful deep learning systems," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 477–487.

[29] D. Cheng, C. Cao, C. Xu, and X. Ma, "Manifesting bugs in machine learning code: An explorative study with mutation testing," in *2018 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2018, pp. 313–324.

[30] A. Aggarwal, P. Lohia, S. Nagar, K. Dey, and D. Saha, "Black box fairness testing of machine learning models," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019, pp. 625–635.

[31] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 2019, pp. 1039–1049.

[32] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *CoRR*, vol. abs/1904.13195, 2019.

[33] Y. Chen, Z. Wang, D. Wang, Y. Yao, and Z. Chen, "Behavior pattern-driven test case selection for deep neural networks," in *IEEE International Conference On Artificial Intelligence Testing*. IEEE, 2019, pp. 89–90.

[34] Z. Li, X. Ma, C. Xu, J. Xu, C. Cao, and J. Lu, "Operational calibration: debugging confidence errors for dnns in the field," in *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2020, pp. 901–913.

[35] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *IEEE Trans. Software Eng.*, vol. 32, no. 9, pp. 733–752, 2006.

[36] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. L. Traon, "Comparing white-box and black-box test prioritization," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 523–534.

[37] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*. ACM, 2009, pp. 201–212.

[38] D. D. Nardo, N. Alshahwan, L. C. Briand, and Y. Labiche, "Coverage-based test case prioritisation: An industrial case study," in *Sixth IEEE International Conference on Software Testing, Verification and Validation*. IEEE Computer Society, 2013, pp. 302–311.

[39] C. Fang, Z. Chen, K. Wu, and Z. Zhao, "Similarity-based test case prioritization using ordered sequences of program entities," *Softw. Qual. J.*, vol. 22, no. 2, pp. 335–361, 2014.

[40] W. Sun, Z. Gao, W. Yang, C. Fang, and Z. Chen, "Multi-objective test case prioritization for GUI applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1074–1079.

[41] K. Zhai, B. Jiang, and W. K. Chan, "Prioritizing test cases for regression testing of location-based services: Metrics, techniques, and case study," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 54–67, 2014.

[42] J. Chen, Y. Bai, D. Hao, Y. Xiong, H. Zhang, L. Zhang, and B. Xie, "Test case prioritization for compilers: A text-vector based approach," in *2016 IEEE international conference on software testing, verification and validation*. IEEE, 2016, pp. 266–277.

[43] J. Chen, Y. Bai, D. Hao, Y. Xiong, H. Zhang, and B. Xie, "Learning to prioritize test programs for compiler testing," in *2017 IEEE/ACM 39th International Conference on Software Engineering*. IEEE, 2017, pp. 700–711.

[44] J. Chen, G. Wang, D. Hao, Y. Xiong, H. Zhang, L. Zhang, and X. Bing, "Coverage prediction for accelerating compiler testing," *IEEE Transactions on Software Engineering*, 2018.

[45] J. Chen, "Learning to accelerate compiler testing," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018, pp. 472–475.

[46] J. Chen, J. Patra, M. Pradel, Y. Xiong, H. Zhang, D. Hao, and L. Zhang, "A survey of compiler testing," *ACM Computing Surveys*, vol. 53, no. 1, pp. 1–36, 2020.

[47] Y. Lou, D. Hao, and L. Zhang, "Mutation-based test-case prioritization in software evolution," in *26th IEEE International Symposium on Software Reliability Engineering*, 2015, pp. 46–57.

[48] D. Shin, S. Yoo, M. Papadakis, and D. Bae, "Empirical evaluation of mutation-based test case prioritization techniques," *Softw. Test. Verification Reliab.*, vol. 29, no. 1-2, 2019.

[49] Z. Wang, M. Yan, J. Chen, S. Liu, and D. Zhang, "Deep learning library testing via effective model generation," in *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 788–799.

[50] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 Science and Information Conference*. IEEE, 2014, pp. 372–378.

[51] D. Pruthi, B. Dhingra, and Z. C. Lipton, "Combating adversarial misspellings with robust word recognition," in *Proceedings of the 57th Conference of the Association for Computational Linguistics*, 2019, pp. 5582–5591.

[52] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2018, pp. 31–36.

[53] J. Ebrahimi, D. Lowd, and D. Dou, "On adversarial examples for character-level neural machine translation," in *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, 2018, pp. 653–663.

[54] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, "Black-box generation of adversarial text sequences to evade deep learning classifiers," in *IEEE Symposium on Security and Privacy Workshops*. IEEE Computer Society, 2018, pp. 50–56.

[55] S. Eger, G. G. Sahin, A. Rücklé, J. Lee, C. Schulz, M. Mesgar, K. Swarnkar, E. Simpson, and I. Gurevych, "Text processing like humans do: Visually attacking and shielding NLP systems," *CoRR*, vol. abs/1903.11508, 2019.

[56] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in *13th IEEE International Conference on Software Testing, Validation and Verification*. IEEE, 2020, pp. 74–84.

[57] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering*. IEEE, 2018, pp. 100–111.

[58] J. Dai and Q. Xu, "Attribute selection based on information gain ratio in fuzzy rough set theory with application to tumor classification," *Applied Software Computing*, vol. 13, no. 1, pp. 211–221, 2013.

[59] A. Grotov and M. de Rijke, "Online learning to rank for information retrieval: SIGIR 2016 tutorial," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2016, pp. 1215–1218.

[60] C. Moreira, P. Calado, and B. Martins, "Learning to rank for expert search in digital libraries of academic publications," *CoRR*, vol. abs/1302.0413, 2013.

[61] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[62] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[63] "Mnist," Accessed: 2020. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[64] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu, "Transfer feature learning with joint distribution adaptation," in *2013 IEEE International Conference on Computer Vision*, 2014.

[65] "Coil-20," Accessed: 2020. [Online]. Available: https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php

[66] "Pie," Accessed: 2020. [Online]. Available: http://www.cs.cmu.edu/afs/cs/project/PIE/MultiPie/Multi-Pie/Home.html

[67] "Driving," Accessed: 2020. [Online]. Available: https://udacity.com/self-driving-car

[68] X. Li and D. Roth, "Learning question classifiers," in *19th International Conference on Computational Linguistics*, 2002.

[69] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, June 2011, pp. 142–150.

[70] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, "Contributions to the study of sms spam filtering: New collection and results," in *Proceedings of the 11th ACM Symposium on Document Engineering*. Association for Computing Machinery, 2011, p. 259–262.

[71] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," *Trans. Assoc. Comput. Linguistics*, vol. 7, pp. 625–641, 2019.

[72] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proceedings of the 11th International AAAI Conference on Web and Social Media*, ser. ICWSM '17, 2017, pp. 512–515.

[73] "Kddcup99," Accessed: 2020. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[74] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *5th International Conference on Learning Representations*, 2017.

[75] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *IEEE European Symposium on Security and Privacy*, 2016, pp. 372–387.

[76] "Keras," Accessed: 2020, https://keras.io/.

[77] J. Chen, Y. Lou, L. Zhang, J. Zhou, X. Wang, D. Hao, and L. Zhang, "Optimizing test prioritization via test distribution analysis," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 656–667.

[78] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proceedings of the International Symposium on Software Testing and Analysis*. ACM, 2000, pp. 102–112.

[79] F. Wilcoxon, S. Katti, and R. A. Wilcox, "Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test," *Selected tables in mathematical statistics*, vol. 1, pp. 171–259, 1970.

[80] M. Alzantot, B. Balaji, and M. B. Srivastava, "Did you hear that? adversarial examples against automatic speech recognition," *CoRR*, vol. abs/1801.00554, 2018.

[81] M. Cissé, Y. Adi, N. Neverova, and J. Keshet, "Houdini: Fooling deep structured visual and speech recognition models with adversarial examples," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017, pp. 6977–6987.