To Appear in 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)

Aster: Encoding Data Augmentation Relations into Seed Test Suites for Robustness Assessment and Fuzzing of Data-Augmented Deep Learning Models

Haipeng Wang[1], Zhengyuan Wei[1], Qilin Zhou[1], Bo Jiang[2], and W. K. Chan[1,*]
[1]City University of Hong Kong, Hong Kong, China
[2]State Key Laboratory of Software Development Environment, School of Computer Science and Engineering,
Beihang University, Beijing, China
haipewang5-c@my.cityu.edu.hk, zywei4-c@my.cityu.edu.hk, qlzhou4-c@my.cityu.edu.hk, jiangbo@buaa.edu.cn,
wkchan@cityu.edu.hk
*corresponding author

Abstract—Data-augmented deep learning models are widely used in real-world applications. However, many state-of-the-art loss-based or coverage-based fuzzing techniques fail to produce fuzzing samples for them from many seeds. This paper proposes *Aster*, a novel technique to address this problem to enhance their fuzzing effectiveness for deep learning models trained with multi-sample data augmentation methods. *Aster* formulates a novel reachability-based strategy to encode the insights of every seed's direct and indirect data augmentation relation instances into the replacement seed of that seed systematically. Our experiment shows that *Aster* is highly effective. On average, loss-based and coverage-based fuzzing techniques can generate 166% and 110% more fuzzing samples and reduce 31% and 22% unsuccessful seeds, respectively, after adopting the replacement seeds generated by *Aster* to replace their original seeds. Their improved models also become up to 55% and 40% on average more robust against FGSM and PGD attacks in the experiment.

Keywords–seed generation; fuzzing; testing; neural network; robustness; data augmentation

1. Introduction

Data augmentation (DA) expands and diversifies the original training dataset by applying various transformations to the training samples. This helps deep learning (DL) models improve their generalization by introducing diverse examples through augmented data. Indeed, data-augmented DL models [1], [2] are widely used in many application scenarios, including AI chatbot [3], [4], autonomous driving [5], [6], disease diagnosis [7], [8], biometric recognition [9], [10].

For brevity, we classify DA methods into two board categories: single-sample methods [11], [12] and multi-sample methods [13]–[17]. Single-sample methods generate augmented data from a sample by applying transformations to the sample alone, such as flipping an image. On the other hand, multi-sample methods (e.g., MixUp [13], CutMix [14]) pair a sample with other samples and combine them (e.g., linear interpolations) to create augmented data.

After data augmentation, the quality of the DL models should be evaluated against robustness before deployment. For instance, if a type of augmented data generalizes a DL model, the assessment and testing of the DL model should include the robustness of the same type of augmented data.

One line of research for model evaluation is to conduct fuzzing [18], [19] on a DL model and check the extent of the model to defend the attacks from the fuzzers (e.g., SENSEI [20], Adapt [18], RobOT [19]) from producing fuzzing samples. In general, fuzzing [18], [19] accepts a pool of seed test cases (*seeds* for short) and evolves the seeds to generate fuzzing samples from these seeds. Previous experiments (e.g., [19]) have shown that loss-based or coverage-based fuzzing techniques [18]–[20] can effectively produce many quality fuzzing samples when fuzzing DL models with augmented data generated by single-sample DA methods. We wonder and formulate the following research question:

Do existing loss-based or coverage-based fuzzers remain effective when fuzzing DL models that are augmented with data generated by multi-sample DA methods?

Nonetheless, we are unaware of any literature answering this research question. The first contribution of this paper is to present the *first work* to answer this research question. Specifically, we conduct the *first* controlled experiment to show that the current generation of (state-of-the-art) loss-based or coverage-based fuzzing techniques [18], [19] are **ineffective** in fuzzing DL models augmented by multi-sample DA methods. Moreover, it is the *first* study to produce fuzzing samples for directly assessing the model generalization by multi-sample DA methods.

We have examined the designs of existing fuzzing algorithms. Some fuzzers such as SENSEI [20] and RobOT [19] focus on iteratively perturbing a sample based on the previous states of the sample only (e.g., guided by the loss value between different versions of a sample [19], [20]). For instance, RobOT [19] mutates the sample iteratively to generate a series of sample candidates until the loss of the generated perturbed samples converges. If a generated sample candidate is an adversarial example, RobOT outputs the sample. Some other fuzzers (e.g., DLFuzz [21], Adapt [18] and FilterFuzz [22]) use Neuron Coverage (NC) [23], [24], activation values [25], or their variants [18] achieved by the samples under fuzzing as the guiding information to generate adversarial examples. However, these metrics are formulated independently of whether multi-sample DA methods couple the given samples or their evolving versions in the training process, making them incapable of assuring against the robustness of the DA relation injected by the multi-sample DA methods into the models under test (**MUT**s). Moreover, suppose the fuzzing time budget to search around an individual sample in a fuzzing campaign is limited (e.g., 5 to 10 seconds in [18], [19]). In this case, in one scenario, a fuzzer may fail to generate any fuzzing samples from a seed due to its ineffectiveness in exploring the search space. We argue that using their resulting sets of fuzzing samples to assess the DL models trained with a multi-sample DA method cannot properly assess the robustness of the models against how well the models learn from such augmented data because many seeds ought to have fuzzing samples but these fuzzers simply cannot find any of these samples. Adopting a naive strategy for fuzzing around the whole set of augmented data is very challenging due to the huge size of the training dataset. The model evaluation thus incurs a threat of omitting the robustness assessment surrounding such relation instances.

In this paper, we propose *Aster*, a novel technique to generate a seed test suite from the training dataset that encodes the information of the data augmentation relation for robustness evaluation of the model under test (MUT) trained with the corresponding multi-sample DA method[1]. *Aster* accepts the training dataset as a seed set X. It iteratively evolves the states of the seeds in X. In each iteration, it composes every seed in the current seed set with the other seeds in the same set that are directly paired with the former captured in the data augmentation process when training the MUT. To compose a seed x with its directly paired seeds to evolve the state of x to a new state, *Aster* prioritizes the latter seeds in decreasing similarity between the prediction vectors of x and each latter seed. It composes the prioritized seeds with the current state of x in turn and stops further evolving the state of x against the remaining seeds if the similarity from the initial seed state does not increase further, to balance the diversity introduced by data augmentation and the target of evolving x away from the initial state to produce a fuzzing sample eventually. Through the iterative compositions of the current states of the seed over multiple iterations, the perturbations between x and the directly and indirectly paired seeds are gradually found (in the sense of reachability search over the augmentation relation) and then composed into the current states of x. Thus, each resulting *replacement candidate* originated from a seed in the original seed pool X encodes the insights of relevant data augmentation relation instances [2]. The downstream fuzzers can then explore the encoded insights to generate fuzzing samples.

We evaluate *Aster* on four benchmark datasets and DL models in different complexities, including FashionMnist [26] with MobileNetV2 [27], SVHN [28] with ShuffleNetV2 [29], CIFAR10 [30] with ResNet20 [31] and CIFAR100 [30] with ResNet56 [31] on Mixup [13]. These datasets and models are widely used in DL testing and robustness improvement-related research [18], [19], [21], [25], [32]. The main results show that *Aster* significantly enhances fuzzers to generate more adversarial examples from more seeds and reduces the ratio of unsuccessful seeds, thereby providing a more comprehensive robustness assessment of the DL models under test. Moreover, after finetuning with the resulting fuzzing samples, the DL models can be significantly more resilient against FGSM and PGD attacks.

The main contribution of this paper is threefold: (1) To the best of our knowledge, this paper presents the first empirical study (controlled experiment) to expose the ineffectiveness of the current generation of loss-based and coverage-based fuzzing techniques to attack DL models trained with multi-sample DA methods. (2) It proposes a novel technique, *Aster*. *Aster* is the first seed generation technique to encode the data augmentation relations over multiple samples into the fuzzing seeds. (3) It shows the high effectiveness of *Aster* by comprehensive experiments.

The rest of this paper is organized as follows. Section II revisits the preliminaries of this work. Sections III and IV present *Aster* and its evaluation. Section V reviews closely related works, and we conclude this paper in Section VI.

2. Preliminaries

2.1 DL Model and Adversarial Examples

A DL model f contains a sequence of layers. The first layer is the input layer, which embeds the input samples into an embedding space. The last layer is the output layer, which produces the prediction result of the DL model. All layers in between are called hidden layers.

Training dataset T, validation dataset V, and test dataset D are normally used for standard training and evaluating DL models. Each sample x in each dataset is labeled with a ground truth c_g. Samples in T, V, and D are called *training samples*, *validation samples*, and *test cases*, respectively. A training scheme uses T to train a DL model f and validates f on V. The generated model is then evaluated on the test dataset D. One kind of DL model is the classification model. Such a model f classifies a sample x and outputs a probability vector, denoted by $f(x)$. The index with the highest prediction probability is regarded as the prediction label, denoted as $c = argmax_c(f(x)[c])$.

Adversarial example [33]–[35] are special samples generated by adding perturbations to clean samples to fool

[1] Take Mixup [13] for example. Mixup produces a shuffled version S of the training dataset T and pairs T and S in the same training epoch. It then produces a set of augmented data from the pair of samples with the same index position in both T and S (e.g., producing an augmented data x' by $x' = \lambda T[i] + (1 - \lambda)S[i]$ for $0 \leq i < |T|$ and $0 \leq \lambda \leq 1$). In this case, MixUp produces a data augmentation relation instance $(T[i], S[i])$.

[2] We note that *Aster* does not judge any changes in prediction labels or compare a prediction label to a ground truth label in generating the final replacement candidates.

a DL model to produce misclassification. An adversarial example x' is generated from an input sample x within the boundary ϵ in p-norm distance, denoted by $x' = x + \Delta$, and satisfying two conditions: (1) $\|x - x'\|_p < \epsilon$ and, (2) $argmax_c(f(x')[c]) \neq argmax_c(f(x)[c])$.

Given a dataset $X \in \{T, V, D\}$, the top-1 accuracy of f on X is the ratio of correct classification over the total number of samples in X, which is $\frac{|\{argmax_c(f(x)[c])=c_g|x \in X\}|}{|X|}$. The top-1 accuracies for the three datasets are called training, validation, and test accuracy, respectively. The sequence of adversarial examples generated from datasets T, V, and D are denoted as *training robust dataset*, *validation robust dataset*, and *test robust dataset*, respectively. The corresponding top-1 accuracies of the three robust version datasets are called the training, validation, and test robust accuracy, respectively.

2.2 Coverage- and Loss-based Generation of Fuzzing Samples

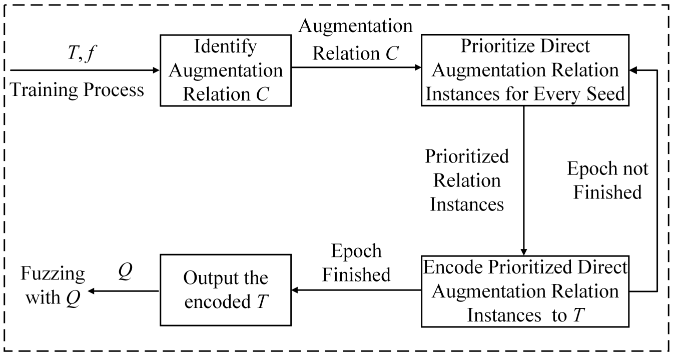
In this section, we revisit two coverage-based and one loss-based fuzzing techniques.

DLFuzz [21] is an early coverage-based fuzzing technique. It mutates the sample x to generate the new sample x' by maximizing the prediction difference between x and x' as well as the neuron coverage [23] for x', while keeping in the x' in the ϵ-bound of x. The objective function is $obj = \sum_{i=0}^{k} c_i - c + \lambda \sum_{i=0}^{m} n_i$, where c is original class label of x, c_i (for $i = 0, ..., k$) is the labels with cofidence lower than c, n_i is the target neuron to be activated by x'.

Adapt [18] is a state-of-the-art coverage-based fuzzing technique. It continuously learns the neuron-selection strategy to guide the fuzzing process iteratively. It designs a vector of 29 Boolean neuron features to characterize the features of neurons in the MUT. Then, it mutates the input test cases x to generate adversarial examples x' by adding the gradient of selected neurons to x and, at the same time, adaptively changes the neuron-selection strategies based on historic neuron-selection strategies and generated adversarial examples in the testing process for further fuzzing process until the assigned time budget expires.

RobOT [19] is the state-of-the-art loss-based fuzzing technique. It generates adversarial examples iteratively until the first-order loss of the generated adversarial example converges. FOL-Fuzz is inspired by the robust training scheme incorporating adversarial examples into the clean training dataset with the objective function $\min_\theta \frac{1}{n} \sum_{i=1}^{n} max_{\|x_i' - x_i\| \leq \epsilon} L(f(x_i'), y_i)$, where θ is the weight matrix of the MUT f, x_i and x_i' are input test case and its adversarial examples correspondingly, ϵ is the bound and y_i is the ground truth for x. The key idea of RobOT is to identify the test cases (around x) with the largest loss.

We also note that in the literature, some fuzzers have adopted pre-trained models with single-sample DA methods (e.g., the pre-trained ImagineNet model [18], [21]) as subjects in their experiments. Their experimental results [18], [19], [21] show that these fuzzers could effectively produce adversarial examples for these MUTs.



T, f
Training Process
Identify Augmentation Relation C
Augmentation Relation C
Prioritize Direct Augmentation Relation Instances for Every Seed
Prioritized Relation Instances
Epoch not Finished
Epoch Finished
Encode Prioritized Direct Augmentation Relation Instances to T
Output the encoded T
Q
Fuzzing with Q

Figure 1. Overview of *Aster*

2.3 Attacker Techniques

The FGSM and PDG methods, as described in [36] and [37], respectively, are commonly used for perturbing samples and evaluating the effectiveness of different robustness improvement methods in experiments. FGSM is a single-step perturbation method that involves adding a small gradient along the loss gradient direction to a sample to produce a perturbed version. PDG is a more refined version of FGSM, which first adds a random gradient to a sample and then iteratively adds perturbations along the current loss gradient.

2.4 Augmentation Relation over Training Samples

DL models trained with empirical risk minimization are sensitive to adversarial examples because the distribution of adversarial examples differs from the distribution of training data. Mixup [13] addresses this problem. For a training sample x with ground truth c_g in a training epoch, Mixup [13] pairs another training sample x_1 with ground truth c_{g1} for x and makes interpolation between them with the formula $x = \lambda \cdot x + (1 - \lambda) \cdot x_1$ and $c_g = \lambda \cdot c_g + (1 - \lambda) \cdot c_{g1}$, where λ picks from Beta(α, α) distribution, and $\alpha \in (0, \inf)$. Then, the generated x and c_g are used to train the model f in that training epoch.

Let T be the original training dataset of the MUT f that is trained with a multi-sample data augmentation method. Suppose that a sample x_i has been paired with the current seed x to produce augmented data to train the MUT. We model all these pairs as a relation denoted by $C : T \to 2^T$, where $x_i \in C(x)$. We refer to C as the ***augmentation relation***. We also refer to the augmentation relation instance between x and x_i as ***direct*** and the sample x_i as a ***DA-paring test case*** of x. If a sample x_j can be reached from x through the compositions of more than one direct augmentation relation instance, we call the relation instance between x and x_j ***indirect***. (For instance, the relation instance between x_j and x is *indirect* if $x_j \notin C(x) \wedge x_j \in C(x_i) \wedge x_i \in C(x)$.)

3. Our Technique: Aster

3.1 Overview

We have presented the overall process of *Aster* in Section 1. Fig. 1 summarizes this overview. Specifically, after obtaining the augmentation relation C, *Aster* iteratively prioritizes and encodes the gradient information of the direct augmentation relation instances of every original seed x into the current state of x's replacement candidate. By so doing, it gradually encodes the gradient information of the corresponding indirect augmentation relation instances of all original seeds into the corresponding replacement candidates. Finally, *Aster* outputs the replacement candidates Q.

3.2 Working Principle

To encode direct augmentation relation instances, our idea is to approximate the encoding of the relation instance by encoding the gradient directions from one seed x to the others paired with x in these augmentation relation instances into a replacement candidate of the seed x. However, for each seed x, as many training epochs may have been applied, many DA-pairing test cases $C(x)$ may have been recorded. Adding all of them to x may perturb x significantly, defying the purpose of keeping the final replacement candidate human-indistinguishable from x.

Our insight is that different samples in $C(x)$ have different distances from (a perturbed version of) x from the MUT's viewpoint. We formulate *Aster* to allow closer samples to encode their required gradient directions among them first, and their perturbed versions are gradually added to the seed x. Thus, more distant samples will contribute smaller perturbation units (shorter gradient directions) to be indirectly or directly added to x. To realize the plan, we use the level of indirectness between two samples in the augmentation relation C as a basis. Samples connected by a shorter composition sequence of direct augmentation relation instances are encoded with the current sample earlier (their distances are deemed shorter). This is realized as encoding the direct relation instances for all seeds in one iteration before proceeding to another. For samples with a direct augmentation relation instance with the current sample, we order them by their similarity to the current sample.

When a fuzzer uses the replacement candidate of x for fuzzing the MUT, it will experience the gradient direction contributions, primarily from the relation instances with DA-pairing test cases of x and secondarily from the relation instances with other DA-pairing test cases of other seeds. By using, for example, the replacement candidate's loss gradient, a fuzzer has the potential to escape from exploring the directions toward diverse augmented data relevant to x.

Fig. 2 illustrates the working principle of *Aster* using six seeds, denoted by ⓐ to ⓕ where C(ⓐ) includes all the other five seeds. We focus on illustrating how the perturbed version of ⓐ encodes the gradient directions from it to samples ⓑ to ⓕ. To simplify the presentation, for brevity, we directly use the symbols ⓐ to ⓕ to represent the perturbed samples of the corresponding seeds and say ⓐ encoding ⓘ to mean that the former sample encodes a perturbation unit of the gradient direction from it to the latter sample into the former sample. In each subfigure, for ease of readers to follow, the distance visually shown in the diagram follows the relative distances from sample ⓐ to other samples (in terms of cosine similarity on their prediction vectors).

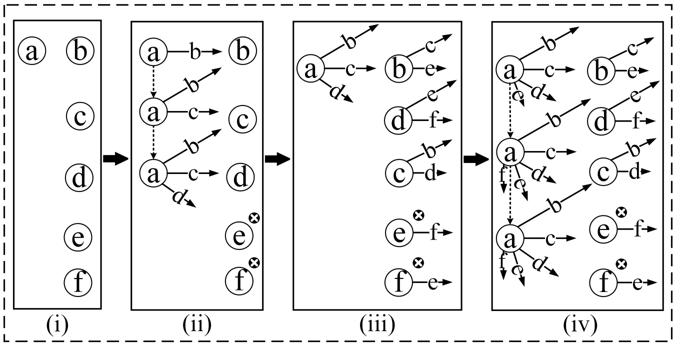


Figure 2. Illustration on Working Principle of *Aster*

Aster starts from the six seeds in subfigure 2(i), which shows the configuration at the start of the current iteration for sample evolution. The seed ⓐ firstly encodes the augmentation relation with ⓑ, then ⓒ, and then ⓓ. As depicted in subfigure 2(ii), the perturbed version ⓐ of the seed gradually evolves to carry the perturbation units of three gradient directions incrementally. When the perturbed version ⓐ further encodes the relation with ⓔ, *Aster* finds the resulting perturbed sample for ⓐ is more dissimilar to the original seed than the one before the current encoding. So, it excludes the sample ⓔ from further direct compositions with ⓐ, which is depicted as ⓔ with a cross symbol in subfigure 2(ii). The sample ⓕ is also excluded because it is more distant from ⓐ than ⓔ when the current iteration starts. After completing the process on ⓐ, *Aster* processes the other samples ⓑ to ⓕ using the same procedure. As depicted in subfigure 2(iii), ⓑ encodes the relations with ⓒ to ⓔ; ⓒ encodes the relations with ⓑ and ⓓ; ⓓ encodes the relations with ⓔ and ⓕ; and ⓔ and ⓕ encode the relations with each other. Note that the distances from the encoding ⓐ to ⓑ–ⓓ may evolve after a round of encoding as depicted in the changes in positions of corresponding symbols between subfigures 2(ii) and 2(iii). In the next iteration to process ⓐ, as depicted in subfigure 2(iv), *Aster* encodes the relations with ⓑ, ⓓ, and ⓒ into ⓐ (i.e., according to their distances from ⓐ). We can see that these samples ⓑ to ⓓ also carry information about the gradient directions of other sample pairs, which are also encoded into ⓐ to different extents (as a way to encode the indirect augmentation relations with respect to ⓐ).

3.3 Algorithm

Algorithm 1 summarizes the *Aster* seed generation algorithm. It accepts three parameters: f is the model under test, X is the training dataset that produces f, and k is the maximum number of steps to decide sample state convergence. At the end of the algorithm, it returns a map Q containing one replacement candidate for each seed test case in X. Note that each test case in Q may or may be an adversarial example of the corresponding seed in T. The downstream fuzzer may use Q instead of X as its seed test cases for fuzzing the model f.

Algorithm 1: The Aster Algorithm

```
Input  : $f \leftarrow$ model under test (MUT)
         $X \leftarrow$ original sequence of seed test cases
         $k \leftarrow$ no. of steps to check state convergence
Output: sequence of seed test cases $Q$
1  $C = \emptyset$, $Q = \emptyset$, $S = \emptyset$, $\mathcal{F} = \emptyset$
2  for $x \in X$ do
3      $C(x)$ = $\text{RELEVANT}(x)$
4      $x' = x + \text{PROCESS}(x, \partial obj/\partial x)$
5      $Q(x)$ = $x'$
6      $S(x)$ = $\text{SIM}(f(x), f(x'))$
7  end for
8  while AVG($\mathcal{F}$) not converged do
9      for $x \in X$ do
10         $u$ = $\langle \text{SIM}(f(Q(x)), f(Q(x_1))) | x_1 \in C(x) \rangle$
11         $k' = 0$, $v = \langle \rangle$
12         for $i \in \text{ARGSORT}(u)$ do
13             if $k' \leq k$ then
14                 $x' = Q(x) + \text{PROCESS}(x, [Q(x) - x] + [Q(C(x)[i]) - x])$    ▷ encoding
15                 if $\text{SIM}(f(x), f(x')) < S(x)$ then
16                     $k' = 0$
17                     $Q(x) = x'$
18                     $S(x) = \text{SIM}(f(x), f(x'))$
19                     $v = v + \langle C(x)[i] \rangle$
20                 else
21                     $k' = k' + 1$
22                 end if
23             else
24                 break
25             end if
26         end for
27         $C(x) = v$
28         $\mathcal{F}(x)$=$\text{AVG}(\langle \text{SIM}(f(Q(x)), f(Q(x_1)) | x_1 \in C(x) \rangle)$
29     end for
30     for $x \in X$ do
31         $y = \text{RAND}(C(x))$
32         $w = \text{CROSSOVER}(C(x), C(y))$
33         $u = \text{AVG}(\langle \text{SIM}(f(Q(x)), f(Q(z))) | z \in w \rangle)$
34         if $u > \mathcal{F}(x)$ then
35             $C(x) = w$
36             $\mathcal{F}(x) = u$
37         end if
38     end for
39 end while
40 return $Q$
```

Algorithm 1 first initializes four maps C, Q, S, and $\mathcal{F}$ in line 1. All of them use the seeds in X as their keys. To know which seeds have been paired with each seed $x \in X$ in training f through the applied multi-sample DA method, we use map C to keep the augmentation relation. Each entry in map S is to keep the similarity score between the prediction vectors of two samples: a seed $x \in X$ and the current replacement candidate x' of x. Map $\mathcal{F}$ is to keep how well the current replacement candidate of each seed $x \in X$ is similar to the current replacement candidates of other seeds that have been paired with this seed x for data augmentation in training the model f. Specifically, it keeps the overall average of the similarity scores on the corresponding prediction vectors for each seed test case $x \in X$.

Algorithm 1 initializes the maps C, Q, and S for the whole seed list X in line 2–7. For each seed $x \in X$, the algorithm retrieves the DA-pairing test cases of x using the function $\text{RELEVANT}(\cdot)$ in line 3. For instance, if the given model under test f is trained with *Mixup* [13], $\text{RELEVANT}(x)$ will return the sequence of samples that have been mixed with x to produce augmented data in training f. If x has never been paired with other seeds to produce augmented data in training f, the function will return a random sequence of seeds as the DA-pairing test cases of x. Line 4 computes the gradient for x backward the objective function $obj = Loss(f(x), c')$ where c' is the one-hot vector to represent the prediction label $c = argmax_c(f(x)[c])$ for x, i.e., $\frac{\partial obj}{\partial x} = \frac{\partial Loss(f(x),c)}{\partial x}$, where the loss function is the cross entropy. The algorithm adopts cross entropy because the entropy term is widely used in many fuzzing techniques [19], [20].

The algorithm then generates a perturbation (within a given bound ϵ) for x using the $\text{PROCESS}(x, \frac{\partial obj}{\partial x})$ function. The $\text{PROCESS}(x, \Delta)$ function accepts two parameters: a seed x and a gradient Δ. It calculates a fraction ϵ of the gradient Δ and returns a clipped fraction of the gradient, i.e., $clipped(\epsilon{\cdot}\Delta)$, to ensure the difference between x and the perturbed version of x does not exceed the bound ϵ. Then, the algorithm adds the clipped fraction of gradient to x to generate a replacement candidate x', where $x' = clipped(x + \epsilon{\cdot}\frac{\partial obj}{\partial x})$ and $\|x' - x\|_p < \epsilon$.

It keeps the replacement candidate x' to $Q(x)$ in line 5. And then, it computes a similarity score between the prediction vectors of the seed x and the replacement candidate x' using $\text{SIM}(\cdot)$ in line 6. Specifically, the $\text{SIM}(a, b)$ function accepts two vectors as input parameters and calculates the cosine similarity [38] between a and b, which is defined as $\text{SIM}(a, b) = \frac{a \cdot b}{\|a\|\|b\|}$. We choose the cosine similarity to measure the correlation between the two prediction vectors due to its simplicity.

Over lines 8 to 39, Algorithm 1 adaptively evolves the replacement candidates in Q. From lines 9 to 29, it iterates over the seed list X. In the iteration for seed $x \in X$, the algorithm aims to reduce the similarity score between the prediction vectors of the replacement candidate $Q(x)$ and the DA-pairing test cases $C(x)$ of the same seed x. Its purpose is to search for a replacement candidate to align with the replacement candidates of these DA-pairing test cases that are best aligned with (i.e., most similar to) the former candidate.

Therefore, in line 10, the algorithm first computes the similarity scores between the prediction vectors of the replacement candidate $Q(x)$ and the replacement candidate $Q(x_1)$ of each DA-pairing test case $x_1 \in C(x)$ and keeps the list of similarity scores into a list u (line 10). It then incrementally evolves the replacement candidate $Q(x)$ (lines 11 to 23). In line 11, the algorithm initializes a counter k' to 0, to record the consecutive number of times the similarity score unable to improve (i.e., for smoothing the checking procedure because of the statistical nature of DL models), and initializes an empty sequence v to store surviving candidates, i.e., the list of DA-pairing test cases can successfully evolve $Q(x)$ to reduce the similarity score between the prediction vectors of $Q(x)$ and x.

In lines 12 to 26, it iterates over u in descending order of similarity score using the $\text{ARGSORT}(\cdot)$ function. These iterations aim to evolve the replacement candidate $Q(x)$ of x by pairing with more similar replacement candidates of the DA-pairing test cases of x first. Note that the function $\text{ARGSORT}(\cdot)$ implemented by *argsort*(.) in *Python numpy* library returns the indexes of the input sequence one by one in sorted order but does not change the input sequence.

If the algorithm exhausts the attempt budget (line 13), it breaks out from the loop (line 24). Otherwise, it aims to search for a better state of the current replacement candidate $Q(x)$ in lines 14 to 19. Similar to line 4, the algorithm at line 14 generates a new state of the replacement candidate x' by computing both a fraction of the gradient from $Q(x)$ to x and another fraction of the gradient from the replacement candidate $Q(C(x)[i])$ of the DA-pairing test case $C(x)[i]$ to x, and adds both fractions to the current replacement candidate $Q(x)$. Specifically, x' is computed as $x' = clipped(Q(x) + \delta\cdot((Q(x){-}x){+}(Q(C(x)[i]){-}x)))$, where $\|x'{-}x\|_p < \epsilon$. (Note that the resulting replacement candidate x' is also clipped to be within the perturbation bound ϵ from x.) Thus, an augmentation relation instance (rather than a perturbation due to an artificial setting such as a loss function irrelevant to the MUT or a subjective criterion on activation values and feature maps) is encoded into x'.

In line 15, if the similarity score between the prediction vectors of the seed x and the new replacement candidate x' is smaller than the best similarity obtained so far, it resets k' to zero as a better state is found, updates $Q(x)$ to keep the latest replacement candidate x', updates $S(x)$ to hold the smaller similarity score, and adds the DA-pairing test case $C(x)[i]$ to v (lines 15–19). Otherwise, it increases k' by 1 to indicate that the remaining attempt budget is reduced (line 21).

After processing the sequence u, in line 27, the algorithm updates the DA-pairing test cases $C(x)$ of x to the sequence of surviving candidates v. So, in the next round of processing the same seed x starting from line 9, the iteration on x can focus on these DA-pairing test cases that can push their replacement candidates toward lower similarity. In line 28, the algorithm updates $\mathcal{F}(x)$ by calculating the mean cosine similarity score between the prediction vectors of $Q(x)$ and the replacement candidate of each DA-pairing test case in the list $C(x)$.

After a round of processing all the seeds in X, over lines 30–39, the algorithm attempts to escape from the local maximum. For each seed $x \in X$, it randomly selects a seed y among the test cases in x's current sequence of DA-pairing test cases $C(x)$ (line 31). It then retrieves the current sequence of DA-pairing test cases $C(y)$ of y, and randomly crossovers the two sequences $C(x)$ and $C(y)$ using the $\text{CROSSOVER}(\cdot)$ function, which returns a new sequence of DA-pairing test cases w for x (line 32). More specifically, given two sequences a and b, the $\text{CROSSOVER}(a, b)$ function firstly generates two random indexes i and j, where $0 < i < |a|$ and $0 < j < |b|$. Then, it appends the sequence $\langle b[j'] | j{<}j'{<}|b|\rangle$ after the sequence $\langle a[i'] | i'{<}i\rangle$, and returns the concatenated sequence. Line 33 calculates the mean cosine similarity score, denoted by u, between the prediction vectors of $Q(x)$ and the replacement candidate of every DA-pairing test case in w. If u is greater than the historic mean similarity stored in $\mathcal{F}(x)$, it updates $C(x)$ to w and $\mathcal{F}(x)$ to u (lines 34–36).

The loop condition at line 8 checks whether the overall mean similarity score for all seeds cannot increase for k consecutive iterations. The algorithm terminates the loop if this is the case.

Finally, Algorithm 1 returns the map Q, which contains the replacement candidates for the corresponding seeds in X.

4. EXPERIMENT

This section reports the evaluation on *Aster*. The implementation code, the datasets, and the model weights are available at https://github.com/KeepSpirit/Aster.

4.1 Research Questions

We aim to answer the following research questions.

RQ1: To what extent is *Aster* effective in boosting the downstream fuzzing techniques to generate more adversarial examples? Is *Aster* effective in enabling these fuzzers to discover adversarial examples that they have difficulties generating adversarial examples from the original seeds? How does *Aster* compare to the state-of-the-art techniques to generate seed test cases? Are there any limitations of existing coverage-based and loss-based fuzzers in fuzzing DL models with multi-sample data augmentation?

RQ2: Is *Aster* more effective than state-of-the-art techniques in boosting the effectiveness of downstream fuzzers to improve the robustness of the models under test?

4.2 Experimental Setup

Environment. We implement *Aster* with the popular machine learning library, Tensorflow 2.4.0. The experiments are conducted on a Windows 10 with an i7-9700 processor, 64GB RAM, and a single NVIDIA GeForce 2080-Ti GPU with 12GB VRAM.

DL model implementations. To evaluate *Aster*, we select four representative DL models, MobileNetV2 [27], ShuffleNetV2 [29], ResNet20 [31] and ResNet56 [31] as our baselines. These baseline DL models are trained with *Mixup* [13] by adopting existing implementations [39]. We choose Mixup as the multi-sample DA method because of its wide adoption in both academic and industry [40]–[45].

Datasets. We choose four benchmark datasets to evaluate *Aster*, including FashionMnist [26], SVHN [28], CIFAR10

TABLE I
DESCRIPTIVE STATISTICS OF BENCHMARKS (BASELINES)

Case	Dataset + Model	Training	Validation	Test
①	FashionMnist+MobileNetV2	89.53	89.16	88.36
②	SVHN+ShuffleNetV2	94.00	93.65	93.47
③	CIFAR10+ResNet20	94.87	91.38	90.28
④	CIFAR100+ResNet56	83.04	69.04	68.66

[30] and CIFAR100 [30]. These four datasets were widely used in experiments to evaluate fuzzing techniques. For test images in each dataset, we randomly divide them into equal halves: one half to serve as the validation dataset and another half to serve as the test dataset in our experiment. Table I summarizes the descriptive statistics of the four baselines DL models trained with Mixup, which shows the case number, the dataset and model, and the training, validation, and test accuracy of each benchmark.

Seed Generators and Fuzzers. We choose three representative test case generation techniques *DLFuzz* [21], *Adapt* [18], and *RobOT* [19] to compare with *Aster* to generate seeds for fuzzers (i.e., as seed generators). We adopt their publicly available code from their Github repositories: *DLFuzz* [46], *Adapt* [47], and *RobOT* [48]. Since these techniques are also fuzzers, we adopt their implementations as the fuzzers to accept the seeds generated by the seed generators.

Hyperparameters. We set the learning rate to 0.001 at the start of training the MUT with Mixup [13], which is reduced by a factor of 10 at every 50 epochs for cases ①–③, and every 100 epochs for case ④. We follow [19] to set up FGSM and PGD: The step sizes of cases ①–② and ③–④ in FGSM and PGD are 0.03 and 0.01, respectively. The fuzzing bound ϵ for fuzzers is 0.05. The single step sizes of cases ①–② and ③–④ in PGD are 0.03/6 and 0.01/6 with the total number of steps 10, respectively. For *DLFuzz* and *Adapt*, we set the activation threshold for neuron coverage to 0.5 and set the fuzzing time budget for cases ①–② to 5 seconds, while 10 seconds for cases ③–④. We follow [19] to set up *RobOT* with $\xi = 10^{-18}$, $k = 5$, $\lambda = 1$ and $iters = 3$, which are described in [19] originally. For *Aster*, we configure Algorithm 1 with p-norm=inf-norm, $k = 3$, $\epsilon = 0.05$, and $\delta = 0.005$. To retrain MUTs, we set training epochs to 40 and the learning rate to $1e^{-4}$ for four cases.

Experimental Procedure. We treat each baseline DL mode as a MUT to answer the above research questions. For each benchmark, we use the whole training samples from the downloaded training dataset as the control group, denoted as the original seed pool P_{Clean}. We input the seed pool P_{Clean} into all remaining seed generators (*Aster*, *DLFuzz*, *Adapt*, and *RobOT*) to output their generated seed pools, denoted as seed pools P_{Aster}, P_{DLFuzz}, P_{Adapt}, and P_{RobOT}, respectively.

We input each seed pool to each fuzzer (*DLFuzz*, *Adapt*, and *RobOT*) to the corresponding MUT with a fuzzing time budget of t seconds where $t \in \{1000, 2000, 3000\}$, which generates an adversarial example dataset A. We count the number of adversarial examples in A generated from each seed. We set the fuzzing time budget following RobOT [19] and extend the duration for studying the feasibility of *Aster*.

We next assess the robustness improvement achieved by different seed generators. We first create a copy B of each set A stated in the paragraph above. We then use the following iterative procedure to produce the retrained model of the MUT: In each iteration, we use the whole set B with the original training dataset to retrain the MUT using the retraining script [48] of *RobOT*. We keep the model with the best validation robust accuracy. To sustain the validation accuracy of the resulting retrained model, suppose its validation accuracy reduces more than 1% compared to the validation accuracy of the MUT. In that case, we randomly delete m samples in the set B and then repeat the iteration, where m equals 10% of the number of samples in A. (Note that the validation accuracy of the resulting retrained model would not reduce more than 1% in our experiments when the size of adversarial examples is smaller than 20000.) Otherwise, the iteration procedure ends. We then evaluate the retrained model's test accuracy and test robust accuracy. We note that the same checking condition is used in retraining a model in the experiment of *RobOT* [19].

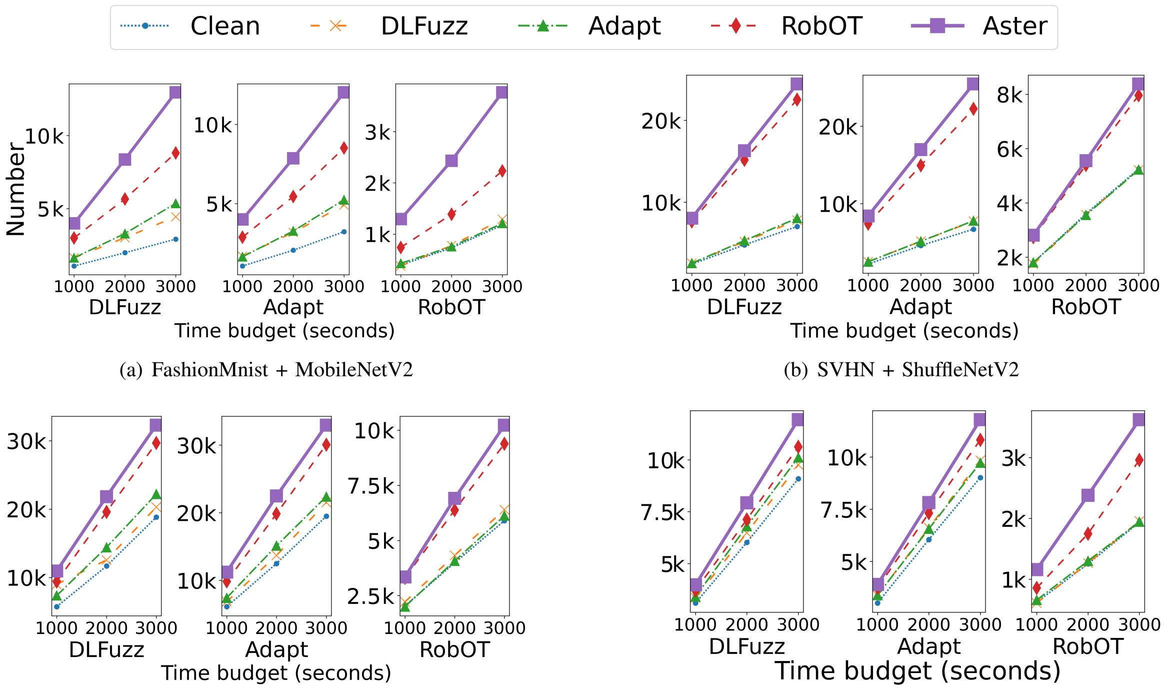
Evaluation and Metrics. To answer RQ1, we measure the number of generated adversarial examples from each seed pool (P_{Clean}, P_{DLFuzz}, P_{Adapt}, P_{RobOT} and P_{Aster}) for each combination of fuzzer and fuzzing time budget. The larger the number of generated adversarial examples, the better the corresponding seed generator.

We conduct the Wilcoxon signed-rank test [49] with Bonferroni correction at the 5% significance level and calculate the effect sizes by Cohen's d [50] to check whether the differences in the number of generated adversarial examples between P_{Aster} and the other seed generators are statistically meaningful. A lower p-value indicates a stronger level of significance. Additionally, the effect size metric categorizes the strength of the relationship between two variables into magnitudes such as very small, small, medium, large, very large, and huge [50]. If the effect size [50] falls under a low strength level (e.g., small), the difference between the two lists is negligible, despite a significant p-value.

A ***processed seed*** of a fuzzer is a seed used to generate perturbed samples by the fuzzer. An ***unsuccessful seed*** is a processed seed, but the fuzzer cannot generate adversarial examples. Given the seed test cases generated from a seed generator, we also measure the ***unsuccessful ratio*** calculated by dividing the number of unsuccessful seeds by the number of processed seeds by the fuzzer using a given fuzzing time budget. The smaller the ratio, the better the seed generator.

We follow *RobOT* [19] to generate test robust datasets for each MUT f from the corresponding test dataset using the same two attack techniques: FGSM [36] and PGD [37]. The sizes of test robust datasets generated by FGSM for cases ①–④ are 2350, 8878, 3185, and 4411, respectively. These generated by PGD are 3783, 12914, 4837, and 4963, respectively.

For each sample in test robust datasets, if a model predicts a label the same as the ground truth label of the sample, we



(a) FashionMnist + MobileNetV2

(b) SVHN + ShuffleNetV2

(c) CIFAR10 + ResNet20

(d) CIFAR100 + ResNet56

Figure 3. Number of generated fuzzing samples: the x-axis shows the time budget used for three fuzzers, and the y-axis is the number of fuzzing samples generated by the fuzzer. *Aster* outperforms all peer techniques in all plots.

count the sample correctly predicted by the model. We refer to the proportion of the test robust dataset that the model predicts correctly as *test robust accuracy* of the model.

To answer RQ2, we measure the *test robust accuracy improvement* achieved by each seed generator: Recall that each retrained model is retrained with the adversarial examples of the seeds generated from a seed generator. We call the retrained model produced from the seed generator. We subtract the test robust accuracy of each MUT f from the test robust accuracy of each corresponding retrained model f' produced by each seed generator and call the result as the test robust accuracy improvement achieved by the seed generator.

Let the test robust accuracy improvement achieved by a seed generator G be A_G. The ***test robust accuracy improvement ratio*** achieved by a seed generator G is $(A_G - A_{Clean})/A_{Clean}$.

4.3 Results and Data Analysis

Producing More Successful Seeds. Fig. 3 contains four sets of plots, one for each case. In each set of plots, there are three plots from left to right corresponding to the three different fuzzers (DLFuzz, Adapt, and RobOT) serving as seed generators to generate fuzzing samples (adversarial examples). Each plot shows five series of points with different markers and line styles corresponding to the five seed generators (*Clean*, *DLFuzz*, *Adapt*, *RobOT*, and *Aster*). Each series of points shows the numbers of generated fuzzing samples by the fuzzer with three different fuzzing time budgets ($t \in \{1000, 2000, 3000\}$ seconds) using the seed pool generated by the corresponding seed generator. We should note that, for all seed pools, the average time budget used in *Aster* for each seed is lower than the other three seed generators.

Across all 12 plots in Fig. 3, for the same combination of fuzzer (x-asix label) and fuzzing time budget (x-value), seeds

TABLE II
UNSUCCESSFUL RATIO OF PROCESSED SEEDS (LOWER IS BETTER)

Case	Seed Generator	Fuzzer (with $t = 3000$s)		
		DLFuzz	Adapt	RobOT
①	Clean	70.18%	64.88%	65.12%
	DLFuzz	70.02%	62.31%	65.08%
	Adapt	65.50%	62.31%	67.13%
	RobOT	47.91%	44.82%	67.13%
	Aster	**30.38%**	**28.05%**	**36.35%**
②	Clean	58.60%	61.27%	**25.61%**
	DLFuzz	58.26%	60.27%	26.10%
	Adapt	57.93%	57.77%	28.59%
	RobOT	23.42%	26.78%	28.10%
	Aster	**8.01%**	**8.51%**	26.51%
③	Clean	27.70%	17.69%	49.57%
	DLFuzz	27.03%	21.00%	51.68%
	Adapt	22.97%	16.67%	54.53%
	RobOT	5.07%	1.69%	55.29%
	Aster	**0%**	**0%**	**40.09%**
④	Clean	10.37%	9.03%	72.29%
	DLFuzz	10.70%	9.03%	73.92%
	Adapt	9.03%	8.70%	73.09%
	RobOT	3.34%	1.03%	74.07%
	Aster	**0%**	**0%**	**22.74%**

TABLE III
TEST ROBUST ACCURACY IMPROVEMENT ON FGSM TEST ROBUST DATASET

Case	Seed Generator	Fuzzer								
		DLFuzz			Adapt			RobOT		
		1000s	2000s	3000s	1000s	2000s	3000s	1000s	2000s	3000s
①	Clean	32.26	37.23	36.68	30.09	35.19	35.11	38.51	40.13	42.38
	DLFuzz	32.26 (0%)	35.53 (−5%)	38.98 (6%)	38.38 (28%)	43.11 (23%)	44.77 (28%)	40.21 (4%)	40.13 (0%)	44.60 (5%)
	Adapt	35.06 (9%)	39.96 (7%)	39.66 (8%)	34.89 (16%)	39.49 (12%)	44.17 (26%)	33.96 (−2%)	40.68 (1%)	40.13 (−5%)
	RobOT	43.66 (35%)	47.32 (27%)	52.81 (44%)	43.70 (45%)	46.98 (34%)	51.02 (45%)	38.02 (−1%)	40.12 (0%)	42.77 (0%)
	Aster	**52.17 (62%)**	**53.11 (43%)**	**60.94 (66%)**	**51.49 (71%)**	**54.77 (56%)**	**59.49 (69%)**	**43.53 (13%)**	**46.40 (16%)**	**49.51 (17%)**
②	Clean	8.41	8.54	8.46	8.67	8.37	8.45	11.16	13.97	14.69
	DLFuzz	8.19 (−3%)	8.34 (−2%)	8.47 (0%)	8.17 (−6%)	8.23 (−2%)	9.63 (14%)	10.67 (−4%)	12.74 (−9%)	14.32 (−3%)
	Adapt	8.03 (−5%)	8.38 (−2%)	8.71 (3%)	8.64 (0%)	8.31 (−1%)	8.62 (2%)	10.06 (−10%)	12.01 (−14%)	13.69 (−7%)
	RobOT	14.08 (67%)	18.24 (114%)	18.65 (120%)	14.29 (65%)	18.54 (122%)	18.39 (118%)	12.16 (9%)	15.88 (14%)	17.22 (17%)
	Aster	**16.65 (98%)**	**21.83 (156%)**	**26.03 (208%)**	**17.14 (98%)**	**22.26 (166%)**	**27.01 (220%)**	**15.32 (37%)**	**17.97 (29%)**	**20.11 (37%)**
③	Clean	24.62	25.34	28.51	24.30	25.65	27.76	30.11	35.35	40.38
	DLFuzz	23.45 (−5%)	26.00 (3%)	27.28 (−4%)	24.02 (−1%)	26.84 (5%)	27.22 (−2%)	29.86 (−1%)	33.94 (−4%)	37.83 (−6%)
	Adapt	23.99 (−3%)	25.87 (2%)	26.15 (−8%)	22.89 (−6%)	27.38 (7%)	27.94 (1%)	28.95 (−4%)	33.50 (−5%)	37.55 (−7%)
	RobOT	28.10 (14%)	32.03 (26%)	35.48 (24%)	24.17 (20%)	32.06 (25%)	35.01 (26%)	28.41 (−6%)	34.69 (−2%)	39.37 (−3%)
	Aster	**30.08 (22%)**	**39.43 (56%)**	**45.15 (58%)**	**30.99 (28%)**	**40.78 (59%)**	**45.18 (63%)**	**33.34 (11%)**	**41.13 (16%)**	**47.66 (18%)**
④	Clean	21.88	22.29	22.47	21.02	21.70	22.47	21.49	22.53	24.35
	DLFuzz	20.72 (−5%)	21.67 (−3%)	23.85 (6%)	20.83 (−1%)	21.81 (1%)	23.37 (4%)	21.24 (−1%)	22.29 (1%)	23.12 (−5%)
	Adapt	21.40 (−2%)	22.85 (3%)	23.60 (5%)	21.42 (2%)	21.74 (0%)	22.58 (0%)	21.76 (1%)	22.87 (2%)	24.76 (2%)
	RobOT	20.86 (−5%)	23.44 (5%)	24.73 (10%)	21.90 (4%)	23.08 (6%)	25.69 (14%)	22.01 (2%)	23.03 (2%)	24.17 (−1%)
	Aster	**23.49 (7%)**	**26.30 (18%)**	**30.95 (38%)**	**23.60 (12%)**	**26.07 (20%)**	**30.29 (35%)**	**24.51 (14%)**	**27.05 (20%)**	**30.54 (25%)**

output by *Aster* always generates more fuzzing samples than seeds output by *Clean*, *DLFuzz*, *Adapt*, and *RobOT*.

In each plot, the *Clean* series is always the lowest. It almost entirely overlaps with the *DLFuzz* and *Adapt* series in nine plots. In the remaining plots, the *DLFuzz* and *Adapt* series are only slightly higher than the *Clean* series. They are significantly lower than the *RobOT* series in 10 plots. Still, all these series have observable gaps from the *Aster* series.

Across three fuzzers, the averaged numbers of generated fuzzing samples among the four benchmarks using the fuzzing time budget $t = 3000$ for the five seed generators (*Clean*, *DLFuzz*, *Adapt*, *RobOT*, and *Aster*) are as follows: (1) 9462, 10573, 11424, 17908, and 20399 for *DLFuzz*; (2) 9607, 11001, 11283, 17910, and 20544 for *Adapt*; and (3) 3568, 3714, 3625, 5635, and 6498 for *RobOT*, respectively. Comparing the numbers of fuzzing samples generated by *Aster* with *Clean*, *DLFuzz*, *Adapt*, and *RobOT*, we observe that *Aster* wins by a large margin: 116%, 93%, 79%, and 14% improvement for *DLFuzz* as the fuzzer, 114%, 87%, 82%, and 15% improvement for *Adapt* as the fuzzer, and 82%, 75%, 79%, and 15% improvement for *RobOT* as the fuzzer, respectively.

We have also conducted the Wilcoxon signed-rank test between *Aster* and each peer seed generator on the number of fuzzing samples generated by all combinations of fuzzers, benchmarks, and time budgets. All p-values are smaller than 5%, and the effect sizes are at either the large or the very large levels, showing that the number of fuzzing samples generated by *Aster* is larger than all peer seed generators significantly in a statistically meaningful way.

Table II shows the unsuccessful ratio of processed seeds (see *Evaluation and Metrics* in Section 4) for each combination of seed generator and fuzzer with 3000 seconds of fuzzing budget. The smaller the ratio, the better the seed generator. The table contains 24 combinations of benchmarks and fuzzers for each seed generator.

Clean has the highest unsuccessful ratios in 6 out of 12 combinations. *Aster* achieves the smallest ratios in 11 combinations (92%). It generates adversarial examples from every processed seed in 4 combinations (33%); whereas no other seed generators can generate adversarial examples from every processed seed in a combination. On average, *Aster* achieves 16.72% in unsuccessful ratio, and the other four seed generators (*Clean*, *DLFuzz*, *Adapt*, and *RobOT*) achieve 44.36%, 44.62%, 43.78%, and 31.55%, respectively. The improvements of *Aster* over other seed generators are large.

Limitations of Current Generation of Coverage-based and Loss-based Fuzzers. Table II also shows the results of the same technique serving as the generator and fuzzer. *DLFuzz* and *Adapt* (the two coverage-based fuzzing techniques) alone are ineffective in Cases ① and ②, and *RobOT* (the loss-based fuzzing technique) alone is ineffective in all but Case ②, which result in high unsuccessful ratios of processed seeds. Their combinations (i.e., using one technique to produce seeds for another technique to produce fuzzing samples) improve the corresponding unsuccessful ratios in all four cases, where the best combinations are *RobOT+Adapt* (44.82%), *RobOT+DLFuzz* (23.42%), *RobOT+Adapt* (1.69%), and *RobOT+Adapt* (1.03%), respectively. All of them achieve lower unsuccessful ratios than using the three fuzzers in a standalone manner. Still, in Case ①, the high ratio of 44.82% indicates that the model assessment based on how well a DL model defends the attack from the Adapt fuzzing task is inadequate. Moreover, these combinations are all less effective than the corresponding best combination of *Aster* (as the seed generator) and the other fuzzer techniques.

TABLE IV
TEST ROBUST ACCURACY IMPROVEMENT ON PGD TEST ROBUST DATASET

Case	Seed Generator	Fuzzer								
		DLFuzz			Adapt			RobOT		
		1000s	2000s	3000s	1000s	2000s	3000s	1000s	2000s	3000s
①	Clean	27.91	32.78	35.58	28.84	33.23	33.68	30.80	33.84	34.71
	DLFuzz	28.26 (1%)	33.41 (2%)	36.66 (3%)	32.91 (14%)	37.32 (12%)	41.16 (22%)	31.88 (4%)	34.39 (2%)	35.69 (3%)
	Adapt	29.87 (7%)	35.34 (8%)	35.18 (−1%)	33.15 (15%)	35.55 (7%)	39.89 (18%)	27.39 (−11%)	32.25 (−4%)	32.38 (−2%)
	RobOT	37.56 (35%)	42.90 (31%)	49.46 (39%)	40.21 (39%)	45.54 (34%)	49.62 (47%)	32.25 (5%)	35.71 (6%)	36.08 (4%)
	Aster	**46.39 (66%)**	**46.97 (43%)**	**58.21 (64%)**	**45.68 (58%)**	**50.70 (53%)**	**56.46 (68%)**	**37.40 (21%)**	**39.36 (16%)**	**44.57 (28%)**
②	Clean	10.14	10.07	10.24	10.19	10.08	10.19	93.4	10.34	10.72
	DLFuzz	10.25 (1%)	10.18 (1%)	10.42 (2%)	9.97 (−2%)	10.41 (3%)	9.45 (−7%)	9.42 (1%)	10.10 (−2%)	10.55 (−2%)
	Adapt	10.07 (−1%)	10.00 (−1%)	10.40 (2%)	9.85 (−3%)	10.32 (2%)	10.44 (2%)	10.12 (8%)	10.35 (0%)	10.64 (−1%)
	RobOT	11.01 (9%)	12.24 (22%)	13.44 (31%)	10.56 (4%)	12.07 (20%)	12.95 (27%)	9.90 (6%)	10.76 (4%)	11.40 (6%)
	Aster	**12.55 (24%)**	**14.80 (47%)**	**17.96 (75%)**	**12.13 (19%)**	**14.85 (47%)**	**18.92 (86%)**	**11.01 (18%)**	**12.26 (19%)**	**13.28 (24%)**
③	Clean	19.04	20.96	23.98	19.60	21.03	23.73	24.60	32.46	37.07
	DLFuzz	18.83 (−1%)	20.63 (3%)	21.50 (23%)	19.81 (1%)	21.54 (2%)	21.98 (−7%)	24.09 (−2%)	29.01 (−11%)	33.47 (−10%)
	Adapt	19.66 (3%)	21.07 (1%)	22.64 (−6%)	19.56 (0%)	22.27 (6%)	22.49 (−5%)	23.78 (−3%)	29.40 (−9%)	33.93 (−8%)
	RobOT	23.44 (23%)	27.56 (31%)	32.54 (36%)	23.65 (21%)	28.30 (35%)	32.25 (36%)	23.20 (−6%)	31.32 (−4%)	36.65 (−4%)
	Aster	**24.62 (29%)**	**36.08 (72%)**	**45.46 (90%)**	**24.83 (27%)**	**37.25 (77%)**	**45.83 (93%)**	**27.33 (11%)**	**38.04 (17%)**	**46.25 (25%)**
④	Clean	12.53	12.79	13.72	11.91	12.39	13.38	12.39	12.75	13.92
	DLFuzz	12.25 (−2%)	13.32 (4%)	13.58 (−1%)	12.07 (1%)	13.32 (−8%)	13.46 (1%)	12.27 (0%)	12.94 (1%)	13.56 (−3%)
	Adapt	11.91 (−5%)	13.30 (4%)	13.70 (0%)	12.37 (4%)	13.26 (7%)	13.46 (1%)	12.33 (0%)	13.14 (3%)	14.37 (3%)
	RobOT	12.75 (2%)	13.50 (6%)	15.01 (9%)	12.71 (7%)	13.90 (12%)	14.79 (11%)	12.96 (5%)	13.20 (4%)	13.44 (−3%)
	Aster	**13.56 (8%)**	**15.27 (19%)**	**19.24 (40%)**	**13.52 (14%)**	**15.64 (26%)**	**18.11 (35%)**	**15.14 (22%)**	**17.37 (36%)**	**18.91 (36%)**

Answering RQ1

Aster is significantly more effective in enabling fuzzers to generate more adversarial examples than *Clean* and the peer seed generators and enhancing fuzzers to generate adversarial examples from significantly more seeds. Moreover, all peer coverage-based and loss-based fuzzers can sometimes be highly ineffective in producing adversarial examples.

Achieving Higher Robust Improvement. Tables III and IV summarize the test robust accuracy improvement results on the FGSM and PGD test robust dataset, respectively. Each table contains four sections, from top to bottom, for Cases ① to ④. The first two columns show the indexes of the four cases (i.e., Cases ① to ④) and the five seed generators (i.e., *Clean*, *DLFuzz*, *Adapt*, *RobOT*, and *Aster*), respectively. The remaining columns show the nine combinations of three fuzzers (*DLFuzz*, *Adapt*, and *RobOT*) and three fuzzing time budgets ($t \in \{1000, 2000, 3000\}$). Each cell in these nine columns shows a pair of values: the test robust accuracy improvement achieved by the seed generator followed by its test robust accuracy improvement ratio (the latter is shown in parentheses). Take *Aster* in the combination of Case ①, *DLFuzz* as the fuzzer, and the time budget of 1000 seconds in Table III as an example. The test robust accuracy improvement achieved by *Aster* is 52.17%, and the test robust accuracy improvement ratio achieved by *Aster* is 62% (rounded to the nearest whole number). The two tables as a whole contain 72 such combinations for each seed generator.

Only *Aster* is consistently higher than *Clean* and all other seed generators in robust accuracy improvement in all 72 combinations. Moreover, the difference is amplified as the fuzzing budget increases. *DLFuzz*, *Adapt*, and *ROBOT* fail in 35, 35, and 12 combinations, respectively, where they even produce worse retrained models than *Clean* (i.e., negative robust accuracy improvement ratio). On average, *Aster* outperforms *Clean*, *DLFuzz*, *Adapt*, and *RobOT* by 40.43%, 37.79%, 38.89%, and 17.69% in robust accuracy improvement ratio, where the differences are large and significant. Also, the robust accuracy improvement ratios achieved by *Aster* in 66 out of the 72 combinations (92%) are higher than the best of all other seed generators by at least 10% in robust accuracy improvement ratio, and 25 combinations (35%) by at least 30%. *RobOT* is more effective than the two coverage-based techniques (*DLFuzz* and *Adapt*) but is not always the case (e.g., in the combination of Case ①, *RobOT* as fuzzer, and time budget of 3000 seconds in Table III and the combination of Case ④, *RobOT* as fuzzer, and time budget of 3000 seconds in Table IV). In 32 out of the 72 combinations (44%), *RobOT* cannot outperform the best of these two techniques by 10% or more in the test robust accuracy improvement ratio. The overall result shows that Aster significantly outperforms the peer seed generators in model robustness improvement.

Answering RQ2

Aster significantly outperforms the peer techniques in improving the robustness of the retrained models, and generalizes for different benchmarks with multi-sample DA methods.

4.4 Threats to Validity

We evaluate *Aster* on limited combinations of DL models, datasets, fuzzers, test robust datasets, hyperparameters, thresholds to produce retrained models, and multi-sample DA method. We have not evaluated *Aster* on MUTs without applying DA or trained with single-sample DA. We leave the generalization as future work. We adopt the publicly available repository [46]–[48] for peer techniques and those codes may contain bugs unknown to us. We have tested our implementation. The test accuracy of all resulting retrained models is within 1% compared to the MUTs. We observe that *Aster* is more efficient than the peer techniques.

5. Related Work

5.1 Data Augmentation

Data augmentation techniques improve the generalization of DL models and alleviate overfitting to training data. They generate additional samples from the original input samples but without additional labeling effort by adopting some special transformation operations. There are geometric transformations data augmentation (e.g., scaling, rotating, and shearing [1], [51]), region-level data augmentation (e.g., *CutMix* [14]) and pixel-level data augmentation (e.g., *Mixup* [13], and *AugMix* [12]). They mix samples and their labels to produce augmented data. In *Aster*, the seeds composed together are related by the data augmentation relation of the underlying DA method and are specific to the model under test. We are not aware of any existing DL testing techniques using any DA relations.

5.2 Loss-based and Coverage-based Fuzzers

DL systems are vulnerable to adversarial examples, highlighting the need to evaluate their robustness. Many fuzzing testing techniques have been proposed to address this issue [18], [19], [21]–[23], [25], [35], [52]–[56].

The coverage-based fuzzing techniques for testing deep learning models aim to maximize the specific coverage criteria as well as discover the misbehavior of the MUT, which mostly rely on structural coverage criteria to guide their techniques, including the use of the neuron coverage in *DeepXplore* [23], DLFuzz [21], and Adapt [18], the k-multisection neuron coverage and neuron boundary coverage in *DeepGauge* [24] or combining multiple coverage criteria in *DeepHunter* [25]. The loss-based fuzzing techniques [19], [20] perturb seed test cases towards the direction of generating larger errors based on their specific adopted loss function, such as the first-order loss in [19] and the cross-entropy loss in [20]. The mutation-based fuzzing techniques [22], [57] generate mutants from the given seed test case and models via different mutators and kill them, which can be time-consuming for adversarial example generations compared to the other two types. These techniques are not designed for fuzzing DL models trained with multi-sample data augmentations. Our experiment has shown their limitations on these models. *Aster* smooths the gradient of a seed by encoding the gradients observed from the DA relation into the seed, thereby providing a gradient clue for a fuzzer to explore the surrounding regions of the resulting seed.

6. Conclusion

DL models trained with one type of augmented data should be tested before deployment on the same type of augmented data. However, even though multi-sample data augmentation is popular to apply, the robustness of the resulting DL models are often not assessed against the same data augmentation. We have presented the first work that generates a robustness-oriented seed set for the training database. This seed set is encoded with the concrete multi-sample data augmentation relation that has been realized in training the DL models under test, which sets it apart from the datasets generated by all other fuzzers. We have proposed a novel technique *Aster* for this purpose and validated its feasibility and high effectiveness. We have also shown the current generation of coverage-based and loss-based fuzzing techniques cannot always fuzz such DL models effectively. Our work calls for the next-generation of testing techniques to test DL models trained with a multi-sample data augmentation method.

7. Acknowledgments

This research is partly supported by the CityU MF EXT (project no. 9678180).

References

[1] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[2] A. Mumuni and F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, p. 100258, 2022.

[3] Y. Shen, L. Heacock, J. Elias, K. D. Hentel, B. Reig, G. Shih, and L. Moy, "Chatgpt and other large language models are double-edged swords," *Radiology*, p. 230163, 2023.

[4] A. Haleem, M. Javaid, and R. P. Singh, "An era of chatgpt as a significant futuristic support tool: A study on features, abilities, and challenges," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 2, no. 4, p. 100089, 2022.

[5] Y. Huai, Y. Chen, S. Almanee, T. Ngo, X. Liao, Z. Wan, Q. A. Chen, and J. Garcia, "Doppelganger test generation for revealing bugs in autonomous driving software," in *Proceedings of 2023 IEEE/ACM 45nd International Conference on Software Engineering (ICSE)*, 2023.

[6] W. B. Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone, "Reward (mis)design for autonomous driving," *Artif. Intell.*, vol. 316, p. 103829, 2023.

[7] D. K. Sharma, M. Chatterjee, G. Kaur, and S. Vavilala, "Deep learning applications for disease diagnosis," in *Deep Learning for Medical Applications with Unique Data*, D. Gupta, U. Kose, A. Khanna, and V. E. Balas, Eds. Academic Press, 2022, pp. 31–51.

[8] T. Liu, E. Siegel, and D. Shen, "Deep learning and medical image analysis for covid-19 diagnosis and prediction," *Annual Review of Biomedical Engineering*, vol. 24, pp. 179–201, 2022.

[9] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, p. 221, 2017.

[10] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[11] K. Alomar, H. I. Aysel, and X. Cai, "Data augmentation in classification and segmentation: A survey and new strategies," *Journal of Imaging*, vol. 9, no. 2, 2023.

[12] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," *arXiv preprint arXiv:1912.02781*, 2019.

[13] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," vol. abs/1710.09412, 2017.

[14] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019, pp. 6022–6031.

[15] A. F. M. S. Uddin, M. S. Monira, W. Shin, T. Chung, and S.-H. Bae, "Saliencymix: A saliency guided data augmentation strategy for better regularization," in *International Conference on Learning Representations*, 2021.

[16] J. Kim, W. Choo, H. Jeong, and H. O. Song, "Co-mixup: Saliency guided joint mixup with supermodular diversity," in *International Conference on Learning Representations*, 2021.

[17] A. Dabouei, S. Soleymani, F. Taherkhani, and N. M. Nasrabadi, "Supermix: Supervising the mixing data augmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 13 794–13 803.

[18] S. Lee, S. Cha, D. Lee, and H. Oh, "Effective white-box testing of deep neural networks with adaptive neuron-selection strategy," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 165–176.

[19] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, and P. Cheng, "Robot: Robustness-oriented testing for deep learning systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 300–311.

[20] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1147–1158.

[21] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018, New York, NY, USA, 2018, p. 739–743.

[22] Z. Wei and W. Chan, "Fuzzing deep learning models against natural robustness with filter coverage," in *Proceedings of 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, 2021, pp. 608–619.

[23] 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*, 2017, pp. 1–18.

[24] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018, New York, NY, USA, 2018, p. 120–131.

[25] 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*, ser. ISSTA 2019, New York, NY, USA, 2019, p. 146–157.

[26] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[27] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

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

[29] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[30] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Master's thesis, University of Tornoto*, 2009.

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

[32] X. Xie, T. Li, J. Wang, L. Ma, Q. Guo, F. Juefei-Xu, and Y. Liu, "Npc: Neuron path coverage via characterizing decision logic of deep neural networks," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, Apr 2022.

[33] E. D. Cubuk, B. Zoph, S. S. Schoenholz, and Q. V. Le, "Intriguing properties of adversarial examples," 2017.

[34] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[35] D. Ma, T. S. Rosing, and X. Jiao, "Testing and enhancing adversarial robustness of hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, p. 1–1, 2023.

[36] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[37] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," 2017.

[38] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi, "The similarity metric," *IEEE transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[39] S. J. Yann N. Dauphin and J. Ma, "Mixup," 2018. [Online]. Available: https://github.com/facebookresearch/mixup-cifar10

[40] E. Pereira, G. Carneiro, and F. R. Cordeiro, "A study on the impact of data augmentation for training convolutional neural networks in the presence of noisy labels," in *2022 35th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, vol. 1, 2022, pp. 25–30.

[41] H. Bai, R. Sun, L. Hong, F. Zhou, N. Ye, H.-J. Ye, S.-H. G. Chan, and Z. Li, "Decaug: Out-of-distribution generalization via decomposed feature representation and semantic augmentation," in *AAAI Conference on Artificial Intelligence*, 2020.

[42] S. Mohseni, H. Wang, C. Xiao, Z. Yu, Z. Wang, and J. Yadawa, "Taxonomy of machine learning safety: A survey and primer," *ACM Comput. Surv.*, vol. 55, no. 8, dec 2022.

[43] C. Li, J. Cao, and X. Zhang, "Robust deep learning method to detect face masks," in *Proceedings of the 2nd International Conference on Artificial Intelligence and Advanced Manufacture*, ser. AIAM2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 74–77.

[44] H. Shui, H. Li, D. Upadhyay, P. Narayanan, and A. S. Admasu, "Mixup for robust image classification - application in continuously transitioning industrial sprays," in *First Workshop on Interpolation Regularizers and Beyond at NeurIPS 2022*, 2022.

[45] S.-H. Lee, J.-H. Kim, H. Chung, and S.-W. Lee, "Voicemixer: Adversarial voice style mixup," 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. 294–308.

[46] Turned2670, "Dlfuzz," 2018. [Online]. Available: https://github.com/turned2670/DLFuzz

[47] Kupl, "Adapt," 2020. [Online]. Available: https://github.com/kupl/ADAPT

[48] W. J. SmallkeyChen, "Robot," 2021. [Online]. Available: https://github.com/Testing4AI/RobOT

[49] R. F. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[50] C. O. Fritz, P. E. Morris, and J. J. Richler, "Effect size estimates: Current use, calculations, and interpretation." *Journal of experimental psychology: General*, vol. 141, no. 1, p. 2, 2012.

[51] J. Gallier, *Geometric methods and applications: for computer science and engineering*. Springer Science & Business Media, 2011, vol. 38.

[52] Y. Hanmo, W. Zan, C. Junjie, L. Shuang, and L. Shuochuan, "Regression fuzzing for deep learning systems," in *2023 IEEE/ACM 45nd International Conference on Software Engineering (ICSE)*, 2023.

[53] Y. Chenyuan, D. Yinlin, Y. Jiayi, T. Yuxing, H. Li, and L. Zhang, "Fuzzing automatic differentiation in deep-learning libraries," in *2023 IEEE/ACM 45nd International Conference on Software Engineering (ICSE)*, 2023.

[54] M. Li, J. Cao, Y. Tian, T. O. Li, M. Wen*, and S.-C. Cheung*, "Comet: Coverage-guided model generation for deep learning library testing," *ACM Trans. Softw. Eng. Methodol.*, feb 2023, just Accepted.

[55] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "TensorFuzz: Debugging neural networks with coverage-guided fuzzing," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97, 09–15 Jun 2019, pp. 4901–4911.

[56] X. Zhang, J. Liu, N. Sun, C. Fang, J. Liu, J. Wang, D. Chai, and Z. Chen, "Duo: Differential fuzzing for deep learning operators," *IEEE Transactions on Reliability*, vol. 70, no. 4, pp. 1671–1685, 2021.

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