DeepSuite: A Test Suite Optimizer for Autonomous Vehicles

Sihan Xu, Zhiyu Wang, Lingling Fan, Xiangrui Cai, Hua Ji, Siau-Cheng Khoo, and Brij B. Gupta

***Abstract*—Deep learning (DL) brings autonomous vehicles (AVs) close to reality. However, the witness of many safety issues has raised a big concern about the reliability of AVs. To solve this problem, much research has been done to test deep learning-driven AVs. Generally, once a test input is produced, a developer needs to manually check its expected output. However, there often exists massive unlabeled test data (e.g., raw context traces in the real world). It is impractical to manually label all test inputs. Despite some works on automatic generation of test oracles, they are either task-specific or constrained to synthetic inputs. In this paper, we present a general and extensible framework, *DeepSuite*, to mitigate the manual effort of generating test oracles. The intuition behind is that not all test inputs are equally worth labelling. With limited testing budget, it is desirable to label a test suite with high diversity and a reasonable size. Due to the large search space, to optimize such test suites is of great challenge. To address it, *DeepSuite* employs a three-phase optimization method (i.e., selection, crossover, and mutation) to iteratively select representative but non-redundant test suites. Such conflicting profit/cost objectives are attained through a genetic algorithm with a well-defined multi-objective fitness function. In the experiments, we first show that the diversity of tests can be revealed by test criteria. Then, experiments on three widely-used datasets demonstrated the effectiveness of *DeepSuite* in generating test suites with competitive testing coverage and 68.42% smaller size, which greatly improves the data collection efficiency of testing DL-driven autonomous vehicles.**

***Index Terms*—autonomous vehicles, data collection, deep learning testing, genetic algorithm, test suite optimization**

I. Introduction

Autonomous vehicles have experienced spectacular progress in recent years, and are expected to improve road safety and benefit economics. Among technological advancements, deep learning (DL) methods exhibit great power and bring autonomous vehicles close to reality [1]–[6]. Despite the progress, erroneous behaviors often occur in DL-driven autonomous vehicles, which could lead to undesirable consequences. For instance, after colliding with a tow truck, a Tesla vehicle burst into flames[1]. The witness of such safety issues has raised a big concern about the reliability of DL-driven autonomous vehicles. To gain trust from users, it is important to perform systematic testing of DL-driven autonomous vehicles (AVs) before deployment.

S. Xu, L. Fan, Z. Wang, X. Cai, and H. Ji are with TKLNDST, College of Cyber Science, Nankai University, Tianjin 300350, China.

S. Khoo is with School of Computing, National University of Singapore, Singapore.

B. B. Gupta is with National Institute of Technology Kurukshetra, Kurukshetra, India & Asia University, Taichung, Taiwan

Corresponding authors: Lingling Fan and Brij B. Gupta.

[1]https://www.reuters.com/article/us-tesla-russia-fire-idUSKCN1V10BB

The quality of test suites is critical to obtain accurate assessments of the reliability of DL-driven AVs. Generally, once a test input is produced, a developer needs to manually specify or check its expected output. Nevertheless, there often exists massive unlabeled data in the real world, e.g., raw context traces. To test all possible inputs needs huge labeling efforts. Despite some works on automatic generation of test oracles, they are either task-specific [7] or constrained to synthetic inputs [8]–[10]. With limited testing budget, it is desirable to obtain a representative and efficient test suite, so that testers can label and test them first.

In this paper, we present a general and extensible framework, *DeepSuite*, to optimize test suites with small sizes while maintaining high testing adequacy. The basic intuition is that not all test inputs are equally important and valuable. Some test inputs can be redundant and replaced by others. Fig. 1 depicts three test inputs for deep neural networks. It can be seen that compared with t_3, t_2 executes similar functionality to t_1. Given a budget of labeling two test inputs, test suite $T_1 = \{t_1, t_3\}$ exhibits more different behaviors of the model and thus more valuable than test suite $T_2 = \{t_1, t_2\}$.

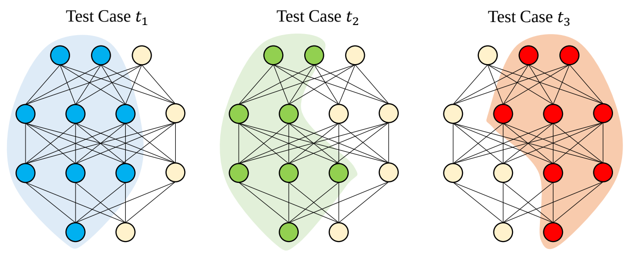

Fig. 1: Test Cases with Different Behaviors

Based on this idea, we propose *DeepSuite* to generate a test suite that is *representative* but *not-redundant* from a pool of unlabeled tests. As the search space is usually very large, to produce such a test suite is a challenging optimization problem. Central to our proposal is the Genetic Algorithm [11]. Starting with an initial population, *DeepSuite* evolves the whole test suite towards three objectives, i.e., maximizing a predefined test criterion, minimizing the test suite size, and minimizing the distance to further improve coverage. We implemented the first objective with five test coverage criteria specialized for DL systems. The reason behind is that it is impossible to detect a defect regarding some neuron activities if these neurons are never activated by tests. Our experiments first prove that well-defined test criteria are correlated with test diversity, and thus can benefit test data selection. Based on this observation, we conduct experiments on five popular test criteria and eight widely-used DL-driven systems. The results show that *DeepSuite* can generate test suites with competitive coverage and 68.42% smaller size.

In summary, we made the following main contributions:

- We investigate and find that well-defined test criteria for DL-driven systems are correlated to the diversity of tests, and thus can benefit test data selection to expose hidden behaviours of DL-driven autonomous vehicles.
- We present *DeepSuite*, an extensible tool to optimize test suites with high testing adequacy but small sizes for DL-driven autonomous vehicles. To our best knowledge, this is the first work on test suite optimization for DL-driven systems with multiple conflicting profit/cost objectives.
- We study the effects of different test criteria and mutation operators on the performance of *DeepSuite*, and find that with metamorphic mutation, *DeepSuite* is capable of achieving higher coverage with limited extra budget.

The rest of the paper is organized as follows. In Section II, we discuss the research works related to this paper. Section III details our three-phase optimization method (i.e., selection, crossover, and mutation). In Section IV, we evaluate the performance of *DeepSuite* on three widely-used datasets. In Section V, we discuss two concerns about a genetic algorithm like *DeepSuite* and the threats to validity as well. Finally, Section VI draws the conclusions and describe future works.

II. Related Work

A. Test Criteria for Deep Neural Networks

Similar to traditional software testing, the quality of test data affects the efficacy of testing. Recently there have been several studies on test criteria for deep learning systems. Pei *et al.* [8] proposed the first coverage criterion for deep learning systems. They highlighted the importance of neuron activities to capture the internal states of deep neural networks (DNNs). A neuron was considered to be activated if the output value is higher than a predefined threshold. Neuron coverage (NC) was defined as the ratio of unique neurons activated by tests. Ma *et al.* [12] designed a set of multi-granularity test criteria for DNNs, including neuron-level and layer-level criteria. Specifically, they proposed k-Multisection Neuron Coverage (KMNC), which divided the output range of each neuron into k sections, and calculated the ratio of covered sections. They also proposed Neuron Boundary Coverage (NBC) and Strong Neuron Activation Coverage (SNAC), which calculated the ratio of covered corner-case regions beyond training data. In addition, they designed Top-k Neuron Coverage (TKNC) [12], which computed the ratio of neurons that have once been one of the most active neurons. Sun *et al.* [13] proposed to apply traditional MC/DC coverage criterion to DNNs. Kim *et al.* [14] proposed a test adequacy criterion named Surprise Adequacy, which was based on the distances of model behaviors between training and test data.

B. Input Synthesis for DL Systems

Test inputs of DL-driven systems can be categorized into two groups, i.e., natural inputs and synthetic inputs. Many studies investigated adversarial example generation to attack deep learning systems [15]. To perform DL testing, Pei *et al.* [8] proposed a white-box testing technique named DeepXplore, which produced inputs to trigger different behaviors of similar DL-driven systems. It perturbed original inputs by gradients towards adversarial examples with high test coverage. Sun *et al.* [16] proposed concolic testing on deep learning systems. Odena *et al.* [17] presented a coverage-guided fuzzing technique that could be used to detect numerical errors and disagreements between DNNs. Similarly, Zhang *et al.* [18] presented a black-box fuzzing technique to find misclassified test inputs for image classifiers. Xie *et al.* [19] implemented *DeepHunter* that leveraged coverage criteria to guide fuzzing. They proposed a metamorphic mutation strategy to preserve the semantics of tests, and automatically detected erroneous behaviors. Tian *et al.* [10] designed DeepTest to perform systematic testing for DL-driven autonomous vehicles. They exploited test coverage to explore different parts of logic, so as to find erroneous behaviors under realistic driving conditions. Shen *et al.* [20] designed an approach named boundary sampling selection to boost mutation testing for DL systems. Despite similar objective (i.e., reduce labelling cost), they constructed test suites to achieve high mutation scores, while *DeepSuite* takes the diversity of test inputs into consideration. In addition, the perturbations of natural inputs is not necessary in *DeepSuite*. The key idea of this paper is to optimize a test suite with high diversity while maintaining small test suite size.

C. Test Prioritization

Traditional Software. Many research efforts have been devoted to test suite optimization for traditional software. The greedy algorithm has been one of the renowned heuristics to optimize test suites [21], [22]. It picked test cases that satisfied a majority of disgruntled requirements and repeated this process until all requirements are fulfilled. The clustering approach was another popular technique to optimize test suites [23], [24]. Researchers typically clustered similar tests into the same group. The basic idea was that similar tests were supposed to exhibit similar behaviors of a target system. Moreover, meta-heuristic algorithms were also explored to optimize test suites [25]–[27]. Inspired by multi-objective regression test selection for traditional software [26], this paper first applies search-based methods on DL-driven AVs.

DL-driven Systems. Test prioritization for DL-driven systems have been investigated by recent works. Shi *et al.* [28] proposed to identify faulty tests from a statistical view of DNNs. Specifically, they exploited cross entropy to prioritize tests. Byun *et al.* [29] presented several strategies to find tests likely to be faulty, including softmax output, Bayesian uncertainty, and input surprise. Moreover, Zhang *et al.* [30] proposed to prioritize tests to generate adversarial examples. Li *et al.* [31] proposed to estimate the performance of DL-driven autonomous vehicles, so as to boost operational testing.

Note that the goal of *DeepSuite* is different from test prioritization. *DeepSuite* highlights the importance of test diversity, and its goal is to optimize a representative and efficient test suite. Although test diversity can be indicative to fault-detection capabilities of test suites, the goals of *DeepSuite* and test prioritization is inherently different.

D. The Oracle Problem

To exercise a test, one needs not only the test input, but also the test oracle that depicts the expected behavior of the test to determine whether the test successes or fails. Nevertheless, it can hardly be assumed that the automated test oracle is always available, especially for DL-driven autonomous vehicles with large input space. To address this problem, one strategy was differential testing that compared the behaviors of a test input when executed on multiple DL systems [8]. Although this strategy could obtain oracles automatically, the requirement of multiple systems with similar functionalities limited its application scope. Another strategy, metamorphic testing, was also a promising solution to tackle the oracle problem [9], [10], [19]. These studies first assumed that there existed a set of inputs that have been labeled correctly. Then, they carefully designed mutation strategies to derive large amounts of synthetic inputs. By assuming such mutation could not affect the semantics of original inputs, they applied metamorphic testing on fuzzing and adversarial example generation. Nevertheless, without complete formal specification, the mutants needed to be manually checked to eliminate false positives.

In the research field of autonomous vehicles, Borkar *et al.* [7] proposed to exploit time slicing to generate oracles automatically. Although this method reduced the manual input of ground truth, it was designed specifically for lane detection, which restricted its application to other tasks such as object detection and tracking [32]. The oracle problem of original and natural inputs still remains the bottleneck of automated testing of DL-driven AVs. In this paper, we mitigate the oracle problem in an alternative way. We take into consideration both the effectiveness and efficiency of natural tests, which are two conflicting profit/cost objectives. We implement an extensible framework named *DeepSuite* and allows metamorphic mutation as an optional setting.

III. APPROACH

In this section, we describe *DeepSuite*, a search-based test suite optimizer for DL-driven autonomous vehicles.

A. Overview of DeepSuite

We first give an overview of the proposed test suite optimizer *DeepSuite*. Fig. 2 depicts the overview of *DeepSuite*. Given a target system and a seeding pool, the search space of *DeepSuite* consists of all possible test suites that can be generated from the seeding pool. To mitigate the huge labelling efforts, *DeepSuite* optimizes a representative and efficient test suite from the seeding pool. To achieve this goal, it employs a search-based optimization method with multiple objectives. It evolves the entire population *iteratively* to find the best individual[2], which is a test suite optimized for the target system. During the evolution, *DeepSuite* first initializes a population with a set of individuals randomly generated from the seeding pool. After that, *DeepSuite* iteratively performs three operators, i.e., *Selection*, *Crossover*, and *Mutation*. The selection operator defines a fitness function to select test suites with high coverage but a small size into further evolution. The crossover operator mates two adjacent candidate test suites to increase the global search ability. The mutation operator allows randomly adding, removing or metamorphic mutating a test case, so as to increase the local search ability of *DeepSuite*. If the fitness cannot be improved anymore or the allocated resources are exhausted, *DeepSuite* stops searching and returns the best candidate, which is the optimized test suite.

Algorithm 1 specifies the main algorithm of *DeepSuite*. The input of the algorithm is a target DL-driven system $\mathcal{N}$ and a pool $\mathcal{P}$ of q unlabelled test inputs. The output is an optimal test suite $\mathcal{T}$. One straightforward way to optimize the test suite is to determine whether a case is selected into the test suite. Without metamorphic mutation, the optimization is a NP complete problem and the algorithmic complexity is 2^q. It gets worse when we consider metamorphic mutation of inputs. Thus, it is impractical to optimize the test suite directly. To address this problem, *DeepSuite* first constructs the initial population G_0 by randomly generating a certain number of individuals, denoted by $G_0 = \{T_1, T_2, \dots, T_n\}$ (Line 2). Each individual is a test suite containing a set of test inputs randomly picked from the pool $\mathcal{P}$, i.e., $T_i = \{t_{i1}, t_{i2}, \dots, t_{im}\}(t_{ij} \in \mathcal{P})$. We use $current\_pop$ to denote the current population in each iteration, and initialize it with G_0 (Line 3). During the evolution, *DeepSuite* evolves the entire population by *iteratively* performing the selection, crossover and mutation operations till the stop condition is satisfied (Line 4). In the selection operation, we design a multi-objective fitness function (obj_1 for coverage, obj_2 for test suite size, and obj_3 for the distance to improve coverage) to measure the effectiveness and efficiency of each individual ind (Lines 5-9). Test suites with higher coverage, smaller sizes, and higher probabilities to improve coverage are assigned higher scores and thus chosen for the next generation denoted by $next\_pop$ (Line 10). To increase the search abilities, *DeepSuite* takes selected individuals as the parents to produce offsprings (Line 12) and performs the mutation operation on these offsprings (Line 13). Specifically, in the crossover operation, *DeepSuite* randomly exchanges a subset of test cases in two adjacent parents (test suites) to produce their offsprings; in the mutation operation, it conducts a two-stage strategy to mutate offsprings at the test-suite and test-case level respectively. At the end of each iteration, offsprings that achieve higher test coverage than parents or the same coverage with smaller sizes are added into the new generation for further evolution (Lines 14-17). *DeepSuite* evolves the entire population till the stop criterion is satisfied and the best individual is selected as the output of the algorithm (Line 18).

B. Fitness Function

The goal of *DeepSuite* is to optimize a test suite with high coverage but a small size from a test pool. However,

[2]We use the terms "individual" and "candidate" interchangeably, because in *DeepSuite* both an individual and a candidate represent a test suite for the target DNN.

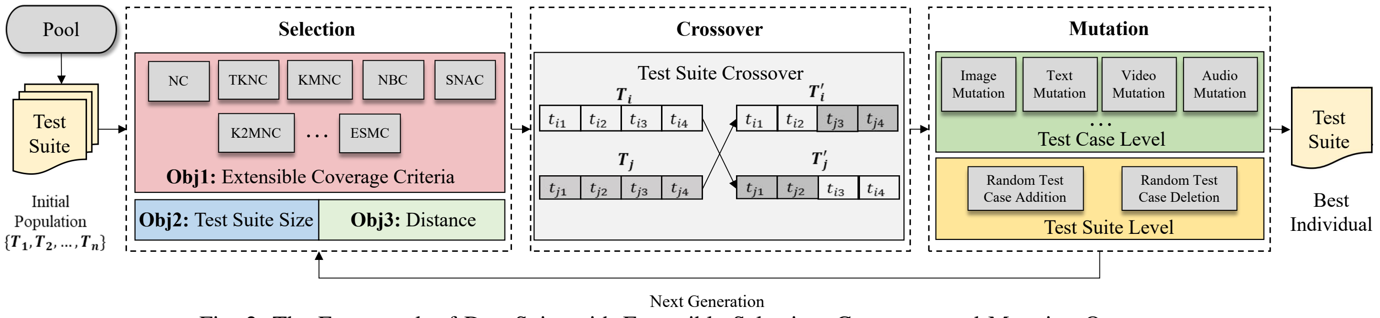
Pool
Test Suite
Initial Population $\{T_1, T_2, ..., T_n\}$
Selection
NC
TKNC
KMNC
NBC
SNAC
K2MNC
...
ESMC
Obj1: Extensible Coverage Criteria
Obj2: Test Suite Size
Obj3: Distance
Crossover
Test Suite Crossover
T_i t_{i1} t_{i2} t_{i3} t_{i4}
T_j t_{j1} t_{j2} t_{j3} t_{j4}
T_i' t_{i1} t_{i2} t_{j3} t_{j4}
T_j' t_{j1} t_{j2} t_{i3} t_{i4}
Mutation
Image Mutation
Text Mutation
Video Mutation
Audio Mutation
...
Test Case Level
Random Test Case Addition
Random Test Case Deletion
Test Suite Level
Test Suite
Best Individual
Next Generation

Fig. 2: The Framework of DeepSuite with Extensible Selection, Crossover, and Mutation Operators

Algorithm 1: Search-Based Test Suite Generation

```
Input: A target DL-driven system $\mathcal{N}$, a set of test inputs $\mathcal{P}$,
       the size of a population $L$
Output: An optimal test suite $T$
1  begin
2      $G_0 \leftarrow initPopulation(\mathcal{P}, L)$ ;
3      $current\_pop \leftarrow G_0$ ;
4      while not reached the stopping criterion do
5          for $ind \in current\_pop$ do
6              $obj_1 \leftarrow Cov(\mathcal{N}, ind)$;
7              $obj_2 \leftarrow Len(ind)$;
8              $obj_3 \leftarrow Dis(ind)$;
9              $ind.fitness \leftarrow (obj_1, obj_2, obj_3)$;
10         $next\_pop, best\_ind \leftarrow Select(current\_pop, L)$;
11         $next\_pop \leftarrow next\_pop \cup \{best\_ind\}$;
12         $offsprings \leftarrow Crossover(next\_pop)$;
13         $offsprings \leftarrow Mutate(offsprings)$;
14         for $ind \in offsprings$ do
15             if $Cov(\mathcal{N}, ind) >$
                 $Cov(\mathcal{N}, best\_ind) \vee Cov(\mathcal{N}, ind) =$
                 $Cov(\mathcal{N}, best\_ind) \wedge Len(ind) <$
                 $Len(best\_ind)$ then
16                 $next\_pop \leftarrow next\_pop \cup \{ind\}$;
17         $current\_pop \leftarrow next\_pop$;
18     $T \leftarrow selectBest(current\_pop)$;
19     return $T$;
```

the search space is usually very large, which is the main challenge of this work. To address this problem, we design a multi-objective fitness function that guides evolution towards the global optimal solution. Specifically, given a feedforward DNN $\mathcal{N}$ and a pool of tests $\mathcal{P}$, we design a fitness function to select test suites that have high test coverage, small sizes, and are likely to improve coverage into further evolution (i.e., with the minimum distance to the activation of new neurons).

Maximizing Testing Adequacy. The first objective is to maximize testing adequacy of test suites. Given a target system, an optimal solution of *DeepSuite* is a test suite that explores different behaviors of the system. To explore various behaviors of a given DNN, we adopt test coverage criteria to measure testing adequacy of test suites [8], [12]. The reason behind is that erroneous behaviors *w.r.t.* a certain neuron can not be exposed unless the neuron is activated. For example, k-multisection neuron coverage (KMNC) is defined as the ratio of sections that have once been covered. Given a neuron n, we first divide the range of values of neuron n in the training set into k sections. Then, the set of values in the i-th section is denoted by V_i^n. If $\omega(t,n) \in V_i^n$, the i-th section of neuron n is covered by the test case t. Formally, given a test suite T, KMNC can be computed by the following equation:

$$Cov_{\text{KMNC}}(T,k) = \frac{\sum_{n \in N} |\{V_i^n | \exists t \in T : \omega(t,n) \in V_i^n\}|}{k \times |N|},$$

where $|N|$ denotes the number of neurons in the target system and k denotes the number of sections in a neuron. Note that *DeepSuite* is an extensible framework that can be combined with any effective test criteria for DL-driven systems.

Minimizing Labeling Effort. The second objective is to minimize labeling effort of test suites. Since automated test oracle is not always available, especially for high-dimensional inputs, it is essential to optimize a representative test suite whose size is not too large. *DeepSuite* achieves this goal by controlling the size of test suites and defraying the cost associated with labeling.

Minimizing the Distance to Improve Coverage. The third objective is to improve test coverage as soon as possible. If multiple candidates (test suites) have the same coverage and sizes, *DeepSuite* computes the nearest distance of each test suite to improve coverage. Specifically, given a test suite, it first records neurons that have not been activated by the test suite. For each inactivated neuron, it finds a test case where the value of this neuron is the closest to the activation threshold, and then computes its distance to activate the neuron. The function $Dis(ind)$ in Algorithm 1 computes the average distance of a test suite to improve test coverage.

Finally, given a test suite T, the fitness of T can be denoted by a tuple, i.e., fitness$(T) = (Cov(T), |T|, Dis(T))$, where $Cov(T)$ represents the test coverage of T on a predefined test coverage criteria, $|T|$ denotes the number of test cases in the test suite, and $Dis(T)$ denotes the average distance of test cases to improve test coverage.

C. Selection Strategy

After initializing a population, *DeepSuite* iteratively selects a set of individuals from the current population into the next generation. Algorithm 2 shows the selection algorithm. We first compute the fitness scores for all individuals in the current population p (Lines 2-7). Then, we iteratively select k individuals from p according to their fitness scores (Lines 9-16). In each iteration, we randomly select k_c individuals from the current population (Line 10). Only the best individual is selected into the next generation. We repeat the selection k times and obtain k individuals into further evolution. We note that there are three objectives in *DeepSuite*, which are not equally important. First, high testing adequacy is the most

Algorithm 2: The Selection Algorithm

```
Input: The current population $p$, the number of selected
       individuals $k$, the number of competitors $k_c$
Output: A population composed of individuals selected for
        the next generation $p_{next}$
1  begin
2      for $ind \in p$ do
3          $obj_1 \leftarrow Cov(ind)$;
4          $obj_2 \leftarrow Len(ind)$;
5          $obj_3 \leftarrow Dis(ind)$;
6          $ind.fitness \leftarrow (obj_1, obj_2, obj_3)$;
7          /*A fitness is a tuple.*/
8      $p_{next} \leftarrow \emptyset$;
9      for $i = 1$ to $k$ do
10         $competitors \leftarrow RandomSelect(p, k_c)$;
11         $sorted\_list \leftarrow SortByDistance(competitors)$;
12         $sorted\_list \leftarrow SortBySize(sorted\_list)$;
13         $sorted\_list \leftarrow SortByCoverage(sorted\_list)$;
14         /*The highest priority is coverage.*/
15         $best\_ind \leftarrow sorted\_list[0]$;
16         $p_{next} \leftarrow p_{next} \cup \{best\_ind\}$;
17     return $p_{next}$;
```

important objective, so that *DeepSuite* first selects test suites with higher test coverage than others. Second, if multiple test suites have the same test coverage, those with smaller sizes are prioritized by *DeepSuite*. The reason behind is that test suites with smaller sizes have fewer redundant test cases than those with the same coverage but larger sizes. Last, if both the testing adequacy and test suite size are the same, *DeepSuite* prioritizes test suites close to the threshold of coverage improvement.

To achieve the aforementioned goal, we sort the candidate list three times with three indicators. Specifically, we first sort the test suites in the ascending order of their distances to improve test coverage (Line 11). Then, we sort the sorted list in the ascending order of test suite sizes (Line 12). Finally, we sort the sorted list again in the descending order of the testing adequacy (Line 13). By this means, test suites with the highest testing adequacy are selected with the highest priority. When multiple test suites have the same testing adequacy, those with smaller sizes are considered have fewer redundant tests, and thus are prioritized. Finally, only if multiple test suites have the same testing adequacy and test suite size, those with smaller distance to improve test adequacy are selected. After sorting, the best individual is the first one in the sorted list (Line 15). We repeat such competition for k times, and finally select k individuals for further evolution (Lines 9-16). We note that *DeepSuite* is an extensible framework that can be combined with other selection operators that are suitable for such a multi-objective optimization problem.

D. Crossover

To enhance the global searching ability, two parent test suites in the current population are mated to produce two offsprings. As illustrated in Fig. 2, we adopt one-point crossover to produce offsprings. Specifically, given two parent test suites T_1 and T_2, we randomly select a crossover point, which is a value between 0 and 1. We use it to denote the ratio θ of the number of test cases that will be swapped. Then we keep the heads of two parent test suites and swap their tails to generate two offsprings. For the reason that each individual is composed of a set of independent test cases, it can be ensured that each offspring is a valid test suite that can be used for testing. In this way, *DeepSuite* is able to pass on useful test cases in the parents to their offsprings.

E. Mutation

To increase the ability to search for optimal solutions, we provide two types of mutation operations, i.e., test-suite level and test-case level mutation. At the test-suite level, *DeepSuite* randomly adds or deletes test cases; at the test-case level, it mutates existing test cases to expose more system behaviors.

1) ***Test-Suite Level:*** Given a test suite as an individual, mutation at the test-suite level randomly adds or deletes test cases to change test suite sizes. For each individual, *DeepSuite* adds a test case randomly selected from the test pool with probability ε. According to the selection strategy, if the test case improve coverage, the individual might be selected into further evolution. Otherwise, the test suite might become obsolete since small-sized test suites with the same coverage are prioritized to filter out redundant cases. Similarly, *DeepSuite* removes a test case from each individual with probability η. When randomly deleting a test case, if it affects test coverage, the test suite might be ignored. Otherwise, if the test coverage remains the same when a test case is removed, the test suite might be selected into the next generation. In this way, the SIZE of test suites changes during the evolution. Combining the selection and mutation operation, *DeepSuite* can reduce the overlap in coverage of different test cases. Generally, it adds a test case only when it improves the entire coverage of the test suite. It removes a test case from a test suite if it has no contributions to test coverage.

2) ***Test-Case Level:*** To further increase the local search capability, *DeepSuite* also allows mutations at the test-case level. Specifically, with probability ξ, it randomly mutates a test case for each individual. When a test case is mutated, we apply domain-specific transformation on it.

Input Transformation. In the following, we use an image as an example input in autonomous vehicles to describe the test-case level mutation. Given an image as a test case, DeepSuite randomly applies image transformation to uncover different behaviors. Specifically, we employ two groups of image transformations: affine transformation and pixel transformation [19]. Affine transformation moves pixels by scaling, shearing, rotation, and translation of original inputs. Pixel transformation changes the values of pixels by means of brightness, contrast, Gaussian noise, and image blurring. The image transformations are designed to find neighbors of original images with similar semantics but different system behaviors, which enhance the local search capability of *DeepSuite*.

Domain-Specific Constraints. In order to ensure semantic similarity of synthetic tests, we apply domain-specific constraints to limit the changes of images. Specifically, conservative parameters are set for image transformations. For example, the amplitude of a rotation is set to less than 15 degrees. For each original image, affine transformation is permitted at most once, and the number of changed pixels is constrained to less than 2 percent of the number of pixels. The image transformation and domain-specific constraints have been proved effective in previous works [19].

We note that *DeepSuite* is an extensible framework that can be applied for general feed-forward DNNs. Mutation at the test-case level can be configured with domain-specific transformations and constraints developed from other research fields. We also note that test-case mutation is optional and extensible. Users can determine whether synthetic tests are permitted in optimized test suites.

F. Stop Condition

The stop condition of *DeepSuite* can be set according to the allocated resources. As a search-based method, *DeepSuite* can evolve towards an optimal solution for an arbitrary length of time, and can be stopped at any time when the allocated resources are exhausted. Theoretically, if running for enough time, *DeepSuite* can converge to an optimal solution, since it always maintains the best solution at each iteration (Line 10 in Algorithm 1). Detailed analysis of convergence can be seen in [33]. In this paper, if the entire population can not evolve for certain generations, we stop searching. To mitigate the problem of local optimal, we run each experiment 30 times with different initialization.

IV. Evaluation

Since *DeepSuite* employs test coverage criteria specialized for DNNs as the guidance of test suite optimization, in this paper, we first investigate the usefulness of five test criteria (i.e., NC, TKNC, KMNC, SNAC, and NBC) in terms of measuring test diversity. Then, we evaluate the performance of *DeepSuite* in generating small-sized test suites with high test coverage. We perform evaluation on a self-driving dataset and two image datasets. Specifically, we address the following research questions:

RQ1: For test selection in *DeepSuite*, are recently proposed test criteria useful to indicate the diversity of test cases?

RQ2: How do hyper-parameters of test criteria affect their effectiveness?

RQ3: Can *DeepSuite* generate small-sized test suites with high testing adequacy?

RQ4: How about the influence of the test-case level mutation on optimized test suites?

A. Dataset

Table I details the datasets and subject models used in this paper. We select three popular and publicly available datasets as the evaluation subject datasets (MNIST[3] and CIFAR-10[4] are two image datasets, and Udacity comes from a self-driving challenge[5]). MNIST consists of 60,000 training inputs and 10,000 testing inputs. Each input is a gray-scale image of hand-written digits. CIFAR-10 is a dataset of general color images with 50,000 training inputs and 10,000 testing inputs.

TABLE I: Datasets and DNN Models

Dataset	Description	DNN Model	#Neurons	Accuracy
MNIST	**Hand-written Digits**	LeNet-1	52	98.3%
		LeNet-4	148	98.3%
		LeNet-5	268	99.1%
CIFAR-10	**Color Images**	VGG-16	2,122	97.4%
		ResNet-20	2,570	91.3%
Udacity	**Self-driving Video**	Dave-orig	1,560	99.9%*
		Dave-norm	1,560	99.9%*
		Dave-drop	1,444	99.9%*

*We report 1-MSE (Mean Squared Error) as the accuracy for the Dave models, because prediction of the steering wheel angle based on the Udacity dataset is a regression task.

The Udacity challenge dataset in Table I is a self-driving car dataset that consists of 101,396 training inputs and 5,614 testing inputs, each of which is a video frame captured by a camera accompanied with steering wheel angle. We study several DNN models for each dataset following previous studies [8], [17], [19]. Specifically, for MNIST, we train three DNN models from the LeNet family and achieve competitive test accuracy (i.e., over 98%) to further conduct a fair comparison. For CIFAR-10, we explore two pre-trained DNN models (i.e., VGG-16 and ResNet-20), which are relatively large-scale models. For the Udacity challenge dataset, like previous works (i.e., *DeepXplore* [8] and *DeepImportance* [34]), we study three pre-trained self-driving models from Nvidia, so as to evaluate the performance of *DeepSuite* on test data selection of autonomous vehicles.

B. Effectiveness of Test Coverage Criteria (RQ1 & RQ2)

1) ***RQ1****:* To provide answers to RQ1, we investigate whether test cases with different functionalities result in different DNN coverage profiles. Since DNN models have no explicit modules of functionalities, test cases belonging to different classes (i.e., with different labels) are considered to exhibit different DL behaviors. However, the prediction of steering wheel angle is a regression task, which results in a continuous value instead of a class label. For this reason, in RQ1&2, we only use five classifiers (i.e., LeNet-1, LeNet-4, LeNet-5, ResNet-20, and VGG-16) to investigate the relationships between diversity and test coverage criteria. Nevertheless, these feed-forward supervised models are inherently the same in terms of representations learning.

For test cases in the same class (i.e., with the same label), we randomly picked 1,000 pairs of test cases within each class, and obtained 10,000 pairs of such test cases from 10 classes. We use C to represent the set of labels in the dataset, and use S_s to denote the set that contains such pairs of test cases, i.e., $S_s = \{(t_i, t_j) | y(t_i) = y(t_j), y(\cdot) \in C\}$ where $y(\cdot)$ represents the label of the test case. We also randomly picked 10,000 pairs of test cases from different classes to compute the average distance for each criterion. We use S_d to denote the set that contains such pairs of test cases, i.e., $S_d = \{(t_i, t_j) | y(t_i) \neq y(t_j), y(\cdot) \in C\}$. The dissimilarities of test cases are measured by Jaccard distance as follows:

$$\delta_{i,j} = 1 - \frac{|g(t_i) \cap g(t_j)|}{|g(t_i) \cup g(t_j)|}, \quad (1)$$

[3]http://yann.lecun.com/exdb/mnist/
[4]https://www.cs.toronto.edu/ kriz/cifar.html
[5]https://github.com/udacity/self-driving-car/tree/master/challenges

TABLE II: Average Jaccard Distances between Test Coverage Profiles

Model	Type	NC_0.5	NC_0.75	TK_1	TK_2	TK_3	KM_10	KM_50	KM_100	KM_200	KM_500	KM_1000	KM_10000
LeNet-1	D_s	0.379	0.146	0.233	0.452	0.382	0.719	0.865	0.883	0.893	0.901	0.906	0.921
	D_d	**0.982**	**0.853**	**0.704**	**0.701**	**0.592**	**0.819**	**0.900**	**0.910**	**0.916**	**0.921**	**0.924**	**0.935**
LeNet-4	D_s	0.526	0.541	0.456	0.445	0.520	0.691	0.832	0.850	0.857	0.863	0.866	0.870
	D_d	**0.855**	**0.948**	**0.816**	**0.760**	**0.662**	**0.800**	**0.882**	**0.893**	**0.898**	**0.901**	**0.902**	**0.905**
LeNet-5	D_s	0.566	0.595	0.406	0.534	0.525	0.630	0.776	0.795	0.804	0.809	0.812	0.815
	D_d	**0.907**	**0.965**	**0.774**	**0.786**	**0.752**	**0.756**	**0.846**	**0.857**	**0.862**	**0.865**	**0.867**	**0.869**
ResNet-20	D_s	0.515	0.756	0.720	0.658	0.611	0.810	0.962	0.980	0.988	0.994	0.996	0.997
	D_d	**0.551**	**0.926**	**0.786**	**0.726**	**0.680**	**0.845**	**0.969**	**0.983**	**0.990**	**0.995**	**0.996**	**0.998**
VGG-16	D_s	0.653	0.737	0.767	0.755	0.738	0.615	0.806	0.836	0.852	0.863	0.867	0.870
	D_d	**0.871**	**0.909**	**0.852**	**0.841**	**0.832**	**0.654**	**0.827**	**0.853**	**0.867**	**0.877**	**0.880**	**0.883**

D_s: average distances between test cases from the *same* class; D_d: average distances between test cases from *different* classes; **TK**: TKNC; **KM**: KMNC

where t_i and t_j are two test cases randomly selected from the test suite, $g(\cdot)$ the function that executes a test case on a target model and obtains a type of coverage profile. We conduct experiments on eight DNN models and five test criteria. However, we observed that for SNAC and NBC, the values in their coverage profiles were almost zeros, which caused their Jaccard distances to be NaNs in most cases. The reason behind is that SNAC and NBC are test criteria for boundary testing (i.e., they only record rare cases beyond the training dataset), For this reason, we only display and analyze the results on NC, TKNC, and KMNC.

For NC, the threshold of neuron activation is set to 0.5 and 0.75. For TKNC, the parameter k is set to 1, 2, and 3. For KMNC, although the parameter k (i.e., the number of value sections) is only set to 1,000 and 10,000 when proposed in [12], we observed that it is difficult to capture the similarities between test cases when the test criterion is too fine-grained. For this reason, we enrich the parameter settings and set k to 10, 50, 100, 200, 500, 1,000, and 10,000.

Result. Table II shows the average Jaccard distances between the coverage profiles of test cases. We use D_s and D_d to denote average distances between test cases from the *same* and *different* classes, respectively. In Table II, a small value implies that test cases are similar to each other in terms of their test coverage profiles. To distinguish between values *w.r.t* different types, distances between test cases from *different* classes are in bold. Overall, it can be seen that given a target DNN model and a test coverage criterion, the average Jaccard distances between test cases from the same classes are smaller than those between test cases from different classes. According to Equation 1, these results show that test cases within the same class have similar coverage profiles compared to test cases from different classes, which indicates that DNN coverage altogether varies for test cases with different functionalities, and thus can be used to measure the diversity of test cases.

Answer to RQ1: Well-defined criteria, such as NC, TKNC, and KMNC, are correlated to the diversity of test cases, and thus can benefit test data selection for DL-driven systems.

*2) **RQ2**:* To answer RQ2, we further investigate the influence of parameters on the usefulness of test criteria to measure test diversity. To illustrate the affects of hyper-parameters, we use Δ to denote the gap between the average Jaccard distances of test cases from the same and different classes. Specifically,

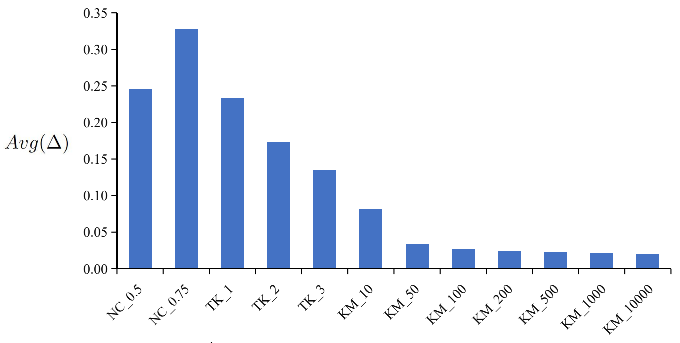

Fig. 3: Averaged Δ for Test Criteria with Different Parameters

we compute Δ as follows:

$$\Delta = \frac{1}{|S_d|}\sum_{(t_i,t_j)\in S_d}\delta_{i,j} - \frac{1}{|S_s|}\sum_{(t_i,t_j)\in S_s}\delta_{i,j}, \qquad (2)$$

where S_s contains pairs of test cases with the same labels, S_d contains pairs of test cases with different labels, and $\delta_{i,j}$ denotes the Jaccard distance between test cases t_i and t_j. According to Equation 1 and 2, a larger Δ indicates that test cases with the same label tend to have similar coverage profiles, compared to those with different labels. If Δ is larger, we expect that the corresponding criterion is more useful to distinguish test cases with different functionalities.

Result. Fig. 3 displays the average Δ for each test criterion on five DNN models. Generally, it can be observed that NC and TKNC have larger $Avg(\Delta)$ than KMNC, which indicates that they are more effective than KMNC in measuring the diversity of test cases in terms of their classes. For NC, it can be seen that $Avg(\Delta)$ increases when the threshold of activation is set to 0.75 rather than 0.5. It indicates that NC is more useful as an indicator of the input-output diversity when the threshold is set to 0.75. For TKNC, where the Top-k activated neurons in each layer are recorded, the parameter k is set to 1, 2, and 3 when the criterion was presented in [12]. From Fig. 3, it can be observed that when k is set to 1, the gap between D_d and D_s is larger compared with the other two settings. For KMNC, it can be observed that when k increases, $Avg(\Delta)$ becomes smaller; when k is set to 1,000 and 10,000 as originally proposed in [12], there is little differences between test cases with the *same* and *different* classes. A possible explanation could be that when k is too large, the value range of a neuron is divided into too many sections. In this case, the criterion is too fine-grained to distinguish test cases with different behaviors.

Answer to RQ2: The parameters of test criteria specialized for DNNs play an important role in their effectiveness to measure the input-output diversity of test cases.

TABLE III: Coverage% (Size) of Test Suites Optimized by Methods without Mutation at Test-Case Level

Model	MTD	NC	KMNC	NBC	SNAC	TKNC
LeNet-1	CL	28.8 (19)	78.5 (198)	1.9 (12)	1.9 (10)	68.2 (20)
	GE	38.5 (12)	98.3 (163)	11.5 (12)	**7.7 (4)**	88.5 (13)
	DS$^-$	**38.5 (6)**	**98.3 (131)**	**11.5 (11)**	**7.7 (4)**	**88.5 (6)**
LeNet-4	CL	57.4 (28)	68.2 (165)	3.4 (25)	4.2 (16)	72.8 (55)
	GE	62.2 (25)	89.1 (183)	5.1 (29)	**7.4 (11)**	83.1 (47)
	DS$^-$	**62.2 (19)**	**89.1 (147)**	**5.1 (20)**	**7.4 (11)**	**83.1 (38)**
LeNet-5	CL	54.1 (59)	77.4 (436)	18.2 (54)	12.6 (44)	65.3 (58)
	GE	**73.9** (55)	83.6 (412)	22.8 (54)	14.6 (43)	76.1 (58)
	DS$^-$	70.5 (**48**)	**83.6 (365)**	**22.8 (50)**	**14.6 (36)**	**76.1 (55)**
ResNet-20	CL	5.3 (157)	71.4(1,564)	9.5 (260)	10.7 (184)	52.9 (578)
	GE	9.6 (132)	**90.2**(1,438)	**13.4** (245)	**14.3** (157)	62.6 (521)
	DS$^-$	**9.6 (98)**	88.0(**1,211**)	12.7 (**227**)	13.0 (**129**)	**62.6 (408)**
VGG-16	CL	12.9 (97)	79.2 (847)	1.3 (275)	5.7 (178)	38.8 (478)
	GE	16.7 (68)	**88.4** (737)	9.1 (242)	15.4 (163)	46.7 (427)
	DS$^-$	**16.7 (56)**	87.9 (**589**)	**9.1 (192)**	**15.5 (157)**	**46.7 (396)**
Dave-orig	CL	36.2 (623)	52.7 (1,925)	0.1 (53)	0.3 (57)	51.2 (843)
	GE	60.5 (453)	63.1 (1,832)	**0.5 (14)**	1.0 (18)	**55.1** (754)
	DS$^-$	**60.5 (436)**	**63.1 (1,739)**	**0.5 (14)**	**1.0 (15)**	53.9 (**691**)
Dave-norm	CL	7.5 (152)	57.4 (2,743)	1.9 (242)	4.7 (185)	8.9 (103)
	GE	**13.8** (133)	**74.3** (2,221)	**4.1** (129)	7.6 (151)	**8.9 (83)**
	DS$^-$	13.6 (**122**)	74.0 (**2,215**)	3.9 (**119**)	**7.8 (114)**	**8.9 (83)**
Dave-dropout	CL	36.6 (352)	46.9 (1,440)	4.3 (69)	4.3 (69)	26.8 (495)
	GE	68.7 (291)	63.2 (1,297)	**6.7** (64)	6.3 (59)	67.7 (397)
	DS$^-$	**68.7 (281)**	**63.5 (1,119)**	6.5 (**49**)	**6.6 (53)**	**67.7 (356)**

MTD: Method; CL: The Clustering Algorithm; GE: The Greedy Algorithm; DS$^-$: DeepSuite without Metamorphic Mutation.

TABLE IV: Coverage% (Size) of Test Suites Optimized by Methods with Mutation at Test-Case Level

Model	MTD	NC	KMNC	NBC	SNAC	TKNC
LeNet-1	BS	38.5 (104)	67.1 (104)	3.8 (104)	1.9 (104)	88.5 (104)
	GA	38.5 (11)	98.5 (103)	38.5 (34)	23.1 (28)	88.5 (9)
	DS	**38.5 (6)**	**98.5 (98)**	**38.6 (29)**	**23.1 (20)**	**88.5 (6)**
LeNet-4	BS	62.9 (157)	64.2 (157)	1.7 (157)	0.0 (157)	46.6 (157)
	GA	62.2 (35)	91.5 (189)	35.1 (104)	14.2 (21)	83.8 (95)
	DS	**62.8 (19)**	**92.9 (142)**	**40.5 (85)**	**15.3 (20)**	**86.5 (36)**
LeNet-5	BS	52.1 (273)	74.9 (273)	10.1 (273)	5.4 (273)	47.8 (273)
	GA	74.6 (74)	83.7 (290)	24.3 (263)	**40.7** (61)	**77.9** (97)
	DS	**76.5 (51)**	**84.7 (235)**	**24.3 (49)**	39.9 (**48**)	77.2 (**54**)
ResNet-20	BS	9.6 (562)	35.1(562)	12.5 (562)	13.8 (562)	62.6 (562)
	GA	14.0 (140)	90.3(1,610)	48.5(1,203)	**74.7** (447)	62.6 (331)
	DS	**14.0 (94)**	**90.9(1,136)**	**49.9 (931)**	72.3 (**370**)	**62.6 (311)**
VGG-16	BS	16.7 (524)	84.6 (524)	11.3 (**524**)	11.9 (524)	46.7 (524)
	GA	20.1 (103)	**88.7** (762)	60.1(1,634)	50.6 (636)	46.7 (455)
	DS	**20.1 (63)**	88.4 (**603**)	**62.3**(1,253)	**50.6 (447)**	**46.7 (314)**
Dave-orig	BS	35.3 (474)	36.2 (474)	0.1 (**474**)	0.1 (474)	49.8 (**474**)
	GA	65.2 (437)	67.5 (1,846)	47.9(1,495)	1.9 (30)	53.3 (819)
	DS	**65.5 (412)**	**71.2 (1,713)**	**47.9**(1,121)	**1.9 (22)**	**58.2** (796)
Dave-norm	BS	13.8 (589)	35.6 (**589**)	3.6 (**589**)	4.1 (589)	8.9 (589)
	GA	16.1 (135)	81.6 (2,396)	55.2(1,703)	**22.7** (338)	42.8 (658)
	DS	**16.1 (106)**	**83.2** (2,052)	**56.1**(1,352)	22.5 (**247**)	**42.8 (472)**
Dave-dropout	BS	68.7 (485)	25.2 (**485**)	6.7 (**485**)	6.7 (485)	22.1 (485)
	GA	78.2 (273)	74.3 (1,536)	54.4(1,463)	15.7 (**111**)	39.4 (560)
	DS	**79.4 (267)**	**81.5** (1,472)	**54.4**(1,315)	**17.4** (113)	**72.5 (339)**

MTD: Method; BS: Boundary Sampling; GA: The Gradient-based Algorithm; DS: DeepSuite with Metamorphic Mutation.

C. *Performance of DeepSuite (RQ3)*

From RQ1&RQ2, it can be observed that test coverage altogether varies for test cases with different functionalities, and thus can approximate the diversity of test cases. Based on this observation, to provide answers to **RQ3**, we conduct experiments on these five test criteria and eight DNN models as related works [8], [19]. We evaluate the performance of *DeepSuite* from two aspects: test adequacy and test suite size. Typically, our evaluation should compare the performance of *DeepSuite* with existing related works. However, to the best of our knowledge, this is the first study to achieve high test coverage with small-sized test suites for DL-driven systems. Therefore, to conduct a comparative study, we self-implemented the following two methods from traditional software to compare with DS$^-$ (*DeepSuite* without metamorphic mutation). **The greedy algorithm (GE) [22]**, based on the renowned reduction-based heuristics, repeatedly selects a test that increases test coverage most until it cannot be improved anymore. **The clustering algorithm (CL) [23]** groups test inputs with similar coverage information into the same cluster, and selects a test from each cluster to form the test suite. If a test is found to be a failure, test inputs in the same cluster with it will also be added into the test suite. We compare DS$^-$ with GE and CL, since test-case level mutation is not permitted in these methods. The results can be seen in Table III.

In addition, to evaluate the performance of *DeepSuite* with metamorphic mutation (denoted by DS), we also implement two state-of-the-art methods that allow mutation testing to compare with *DeepSuite* (the results can be seen in Table IV). The **gradient-based strategy (GA) [8]** mutates original inputs towards high test coverage. Specifically, for each test, the mutation stops when test coverage cannot be improved within a predefined number of steps. The **boundary sampling strategy (BS) [20]** selects tests close to the decision boundary, so as to come up with small-sized subsets with high mutation scores. To ensure the correctness of our implementation, we strictly follow the implementation details accordingly and cross-validate the implementation with co-authors.

As for the clustering algorithm, we use the k-means algorithm to cluster test inputs, and empirically set k to 10. Given an uncovered neuron, the gradient-based algorithm tries to modify an input such that the coverage is increased. For each original test, we empirically set the maximum number of trials to 80. In *DeepSuite*, we initialize the size of each population to 80, and each individual is initially composed of 6 test inputs randomly picked from a pool of unlabeled tests. The probability of mutation operators including addition, deletion, and mutation of a test case is set to 0.3. To conduct a fair comparison, we use the same constraints (i.e., the maximum number of changed pixels and the maximum value of changes) for DS and GA. The details of constraints can be seen in Section IV-D. To counter the randomness of these strategies, we randomly select 5,000 tests from original tests to form pools of unlabeled tests, and run each experiment for 30 times. Based on the observations in RQ1 and RQ2, to achieve better performance, we set the threshold value for NC to 0.75, compute the most active neurons in each layer for TKNC (i.e., set $k = 1$), and divide the range of values for each neuron into 10 equal sections (i.e., set $k = 10$) for KMNC.

As a search-based algorithm, DeepSuite can keep searching towards an optimal solution for an arbitrary length of time, and stops at any time when allocated resources are exhausted. The longer *DeepSuite* evolves, the better results will be obtained. However, a developer might not be willing to wait for too long to obtain results. A reasonable trade-off between effectiveness and efficiency is in need. To conduct a fair comparison, we conduct all the experiments with bounded computational time. Specifically, for each model, we empirically set the timeout to be 5 seconds per neuron. The intuition behind is that the computational complexity of test coverage is linearly correlated with the number of neurons. If *DeepSuite* can generate test suites with high coverage but small sizes, we expect that there are few duplication in generated test suites, which makes the resulting test suites worth labeling.

Result. Table III and IV summarize average test coverage and sizes of test suites optimized by methods without and with metamorphic mutation, respectively. The highest coverage and the smallest test suite size are highlighted in bold. If there exist more than one highest coverage, only the coverage accompanied with the relatively small test suite size is highlighted. It can be seen that compared with CL and GE, *DeepSuite* without metamorphic mutation (denoted by DS^-) achieves the highest coverage in 30 out of 40 cases. Compared with BS and GA, *DeepSuite* with metamorphic mutation (denoted by DS) achieves the highest coverage in 35 out of 40 cases. The high test coverage achieved by DS^- and DS indicate that *DeepSuite* is able to exhibit more different behaviors than those optimized by other methods. For instance, the improvement of NC coverage indicates the activation of new neurons that have not been activated by other methods. It can also be observed from Table III that DS^- generates the smallest test suites among DS^-, CL, and GE in all cases (3 cases have the same size with GE). From Table IV it can be seen that DS generates the smallest test suites in 32 out of 40 cases. Combining the test coverage and test suite size, it can be found that *DeepSuite* generated test suites with high coverage but small sizes. The results in Table III and IV imply that *DeepSuite* has the ability to save the labeling effort while maintaining testing diversity.

From Table IV it can be found that in some cases, although the test suites optimized by DS are not among the smallest test suites, their coverage is much higher than the competitors. For example, with 18 additional tests, DS improves the NBC coverage on LeNet-1 by 236% times compared with DS^- and GE. Moreover, it can also be observed that the coverage of test suites optimized by CL is the lowest among CL, GE, and DS^-. A possible reason could be that CL only picks test cases within the same cluster with a failing case, while it ignores other test cases that may improve coverage and diversity. Although both BS and *DeepSuite* aim at generating representative and small-sized test suites, BS selects tests according to their distance to the decision boundary. Therefore, it can be seen that BS generates same-sized test suites for five metrics, and achieves lower test coverage than *DeepSuite*. In addition, DS^- achieves competitive coverage with GE, which repeatedly adds tests into test suites until coverage cannot be improved anymore. Nevertheless, the sizes of test suites optimized by DS^- are the smallest among CL, GE, and DS^- in all cases. Similarly, comparing GA and DS, it can be observed that DS achieves competitive coverage with GA, while the test suite size of DS is much smaller than GA. For instance, DS and DS^- only need 6 test cases to reach the maximum NC coverage, while CL, GE, and GA need 19, 12, and 11 test cases, respectively. The reason is that *DeepSuite* takes into consideration both test adequacy and test suite size during evolution. By controlling the size of optimized test suites, *DeepSuite* is capable of saving the labelling effort and achieving high testing coverage as well.

Answer to RQ3: *DeepSuite* can mitigate the oracle problem of testing DL-driven autonomous vehicles by generating small-sized test suites with competitive testing coverage to save the labeling effort.

D. Effects of Mutation Operators in DeepSuite (RQ4)

In *DeepSuite*, we implement two mutation operators to increase the local search capability to find optimal solutions. Mutation at test-suite level, which is mandatory in *DeepSuite*, randomly adds or deletes tests in each individual with a predefined probability; mutation at test-case level, which is optional in *DeepSuite*, allows metamorphic mutation of original tests. In this section, we study the effects of test-case level mutation and provide answers to **RQ4**. Like previous studies [8], [19], we adopt a conservative strategy to preserve the semantics of inputs. Specifically, affine transformation is applied only once; we set the maximum ratio of the number of changed pixels to 0.1, and set the maximum value that a pixel may change to 51 (i.e., 0.2×255).

Result. From Table III and Table IV, it can be observed that DS achieves higher coverage than DS^- in 36 out of 40 cases, and they achieve the same coverage in the rest cases. It is consistent with the intuition that more behaviors of DNN models can be discovered by allowing domain-specific metamorphic mutation of original tests. In the meantime, the test suite sizes of DS are not larger than those of DS^- in 19 out of 40 cases, which indicates that with metamorphic mutation, DS has the capability of achieving high testing coverage with limited extra testing budget.

Combining coverage and test suite size, it can be seen from Table IV that in 15 out of 40 cases, DS achieves higher test coverage than DS^- with no-larger test suites. In 21 out of 40 cases, DS increases the test coverage and the test suite size in the meantime, which indicates that DS activates more neurons by adding more synthetic tests. For instance, DS achieves higher NBC coverage than DS^- by 694% times on LeNet-4, with 65 extra tests. In the rest cases, DS maintains test coverage with no-larger test suites than DS^-. Moreover, the results in Table IV also indicate that with metamorphic mutation, GA and DS achieve higher coverage than methods without metamorphic mutation. Among these methods, GA achieves competitive coverage with DS. Nevertheless, test suites optimized by GA have larger sizes than those optimized by DS in almost all cases.

Since the goal of *DeepSuite* is to mitigate the oracle problem by optimizing test suites that are worth labelling. To better characterize the performance of DS and DS^-, we compute the average coverage reached by per test for each coverage criteria. Fig. 4 illustrates the efficiency of optimized test suites, where the Y-axis indicates the average test coverage achieved by per test generated by different methods. The following results can be observed and inferred from Fig. 4. (1) Both DS and DS^- are capable of optimizing test suites with relatively high coverage but small sizes. Given a test criterion, the average test coverage

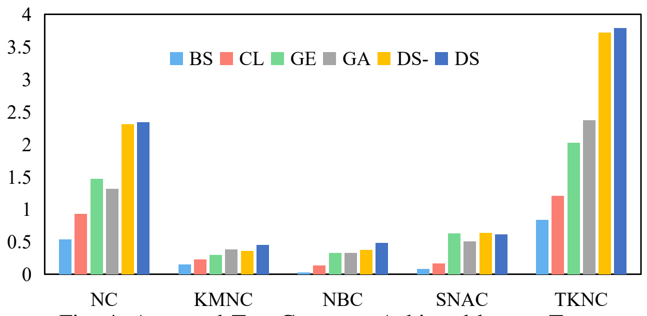

Fig. 4: Averaged Test Coverage Achieved by per Test

achieved by tests optimized by DS is higher than others. It indicates that there are fewer redundant tests in the test suites optimized by *DeepSuite*, and more behaviours can be found with limited labelling efforts. (2) The efficiency of test suites (i.e., coverage per test) generated by DS is slightly better than that of DS$^-$, which is consistent with the intuition that with metamorphic mutation, *DeepSuite* is able to touch more corner cases with limited extra tests. (3) TKNC and NC are easier to be satisfied by fewer test cases compared to KMNC, NBC and SNAC. We note that *DeepSuite* is an extensible tool that can be configured with an arbitrary criterion that is suited to DL-based systems.

Answer to RQ4: With metamorphic mutation, *DeepSuite* is capable of achieving higher test coverage over DL-driven systems with limited extra testing budget.

V. Discussion

A. Local Optimum and Randomness

The genetic algorithm often suffers from the premature convergence, which is caused by an early homogenization of genetic materials in the population. In this case, the generated offsprings are not able to outperform their parents and no valuable exploration can be performed further. To prevent premature convergence, we adopt several strategies to regain genetic variation, including incest prevention, uniform crossover, and increasing population size. We also run each experiment for 30 times and report the average results to counter the randomness. Specifically, first, *DeepSuite* is configured with lots of randomness during the entire evolution, from initialization to mutation. To be specific, the initial population consists of several individuals, where each individual contains a set of test inputs *randomly* picked from the testing pool. After selection, *DeepSuite randomly* exchanges a subset of test cases in two adjacent test suites. In the test-suite level mutation, *DeepSuite randomly* determines to add or remove a test case for each individual, and the test case is picked with *randomness*. In the test-case level mutation, *DeepSuite* allows *random* input transformation within domain-specific constraints for a *random* test case. In the evaluation, we run each experiment for 30 times and compute the average results to counter the randomness. In practical use, users can stop *DeepSuite* at an arbitrary time, and it will maintain the best individual it finds so far. For this reason, to stop and restart *DeepSuite* with random initialization can also help to avoid trapping in a local optimum.

B. The Efficiency and Effectiveness

Another concern about a genetic algorithm like *DeepSuite* is the time cost, i.e., how long it takes to converge into the optimal solution. As *DeepSuite* is an extensible framework that consists of multiple components, the time cost of *DeepSuite* is determined by its configuration. Specifically, users can select or customize a test criterion as the first objective in the fitness function. However, the computational complexity of the criterion has an big impact on the efficiency of *DeepSuite*, because it needs to compute the fitness scores for individuals in each generation. Moreover, there are also some parameters that may have some impacts on the efficiency, such as the initialization and the probability of crossover and mutation. To facilitate evolution, we have implemented some strategies in *DeepSuite*. First, in the selection operator, we design *DeepSuite* to select test suites with high coverage and small sizes. When multiple test suites have the same coverage and sizes, which usually happens when it evolves for enough time, *DeepSuite* is configured to select test suites that are closer to the activation of new neurons (e.g., close to the predefined threshold of activation in NC), so as to improve test coverage and find more behaviors. Moreover, when a test case is randomly added into a test suite or mutated, most test cases remain the same with the previous generation. In this case, only new test cases are executed to calculate the new fitness score.

C. Extensibility

DeepSuite is an extensible framework that can be applied for general feed-forward DNNs, and combined with other selection, crossover, and mutation operators not described in this paper. Specifically, in the selection operator, we use five test coverage metrics, as well as test suite size and distance to form a multi-objective fitness function. Actually, *DeepSuite* can be combined with other fitness functions suited for DNNs. Similarly, although we describe the Tournament selection and one-point crossover in the paper, other selection strategies (e.g., non-dominated sorting) and crossover strategies (e.g., two-point crossover) can also be applied in *DeepSuite*. In the mutation operator, DeepSuite mutates each individual (i.e., a test suite) at two levels: the test-case level and test-suite level. While we use images as example inputs, mutation at test-case level can be configured with domain-specific transformations and constraints for other types of inputs in autonomous vehicles. We also note that mutation at test-case level is optional and extensible, since users may not want to mutate original inputs. Users can determine whether synthetic tests are permitted in optimized test suites.

D. Threats to Validity

Threats that may have an impact on the results of this work include the following two aspects. First, the subject datasets and neural networks in the evaluation could be a threat to validity. However, we have conducted experiments on several datasets and DNN models in different research fields to mitigate it, including two popular image datasets and a widely-used self-driving dataset. Second, when mutation at test-case level is turned on, the metamorphic mutation of original tests are assumed to preserve semantics of tests. However, without formal verification, it is difficult to prove that synthetic inputs are semantically useful to perform testing. We mitigate this issue by applying conservative strategies that only allow human imperceptible perturbations.

VI. Conclusion and Future Work

DeepSuite is a framework to optimize test suites for DL-driven autonomous vehicles; its goal is to generate test suites which exhibit high testing adequacy and maintain small suite size simultaneously. It does not aim to resolve automatic oracle generation problem; instead it aims to mitigate this problem. In this paper, we first conduct correlation analysis to investigate the efficacy of existing test criteria for DL-driven systems, which is of great interest in the field of deep learning testing recently. It is shown that different test criteria may have different impacts on structural testing; well-defined test criteria are correlated to the diversity of test cases, and thus can be useful to guide the data selection process of testing DL-driven autonomous vehicles. Then, the empirical study on the five criteria and eight DL models has clearly demonstrated that the test suites optimized by *DeepSuite* is consistently small but maintain competitive coverage. Moreover, *DeepSuite* can incorporate metamorphic mutation operations to touch more corner cases, improving test adequacy with optimal test suite size. Finally, we note that *DeepSuite* is an *extensible* framework that can be combined with any effective test criteria and mutation strategies tailored to DL-driven autonomous vehicles. The future work includes two aspects. On the one hand, we plan to promote the efficiency of *DeepSuite*, and design better fitness functions to facilitate the data selection process. On the other hand, *DeepSuite* can be applied in many types of inputs in autonomous vehicles other than images, such as videos and radar inputs.

Acknowledgment

This work was supported by National Key Project of China (No. 2020YFB1005700), National Natural Science Foundation of China (No. 62102197, 62002178), NSFC-General Technology Joint Fund for Basic Research (No. U1936206), and Natural Science Foundation of Tianjin, China (No. 20JC-QNJC01730).

References

[1] D. Töpfer, J. Spehr, J. Effertz, and C. Stiller, "Efficient road scene understanding for intelligent vehicles using compositional hierarchical models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 441–451, 2014.

[2] H. Fatemidokht, M. K. Rafsanjani, B. B. Gupta, and C.-H. Hsu, "Efficient and secure routing protocol based on artificial intelligence algorithms with uav-assisted for vehicular ad hoc networks in intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[3] H. Zhu, K.-V. Yuen, L. Mihaylova, and H. Leung, "Overview of environment perception for intelligent vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 10, pp. 2584–2601, 2017.

[4] P. Yang, G. Zhang, L. Wang, L. Xu, Q. Deng, and M.-H. Yang, "A part-aware multi-scale fully convolutional network for pedestrian detection," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[5] Y. Tian, J. Gelernter, X. Wang, J. Li, and Y. Yu, "Traffic sign detection using a multi-scale recurrent attention network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4466–4475, 2019.

[6] F. Mirsadeghi, M. K. Rafsanjani, and B. B. Gupta, "A trust infrastructure based authentication method for clustered vehicular ad hoc networks," *Peer-to-Peer Networking and Applications*, pp. 1–17, 2020.

[7] A. Borkar, M. Hayes, and M. T. Smith, "A novel lane detection system with efficient ground truth generation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 365–374, 2011.

[8] 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.

[9] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 132–142.

[10] 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*, 2018, pp. 303–314.

[11] J. Campos, Y. Ge, G. Fraser, M. Eler, and A. Arcuri, "An empirical evaluation of evolutionary algorithms for test suite generation," in *International Symposium on Search Based Software Engineering*. Springer, 2017, pp. 33–48.

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

[13] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

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

[15] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, and R. Long, "Technical report on the cleverhans v2.1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2018.

[16] 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*, 2018, pp. 109–119.

[17] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," *arXiv preprint arXiv:1807.10875*, 2018.

[18] F. Zhang, S. P. Chowdhury, and M. Christakis, "Deepsearch: Simple and effective blackbox fuzzing of deep neural networks," *arXiv preprint arXiv:1910.06296*, 2019.

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

[20] W. Shen, Y. Li, Y. Han, L. Chen, D. Wu, Y. Zhou, and B. Xu, "Boundary sampling to boost mutation testing for deep learning models," *Information and Software Technology*, vol. 130, p. 106413, 2021.

[21] S. Singh and R. Shree, "A combined approach to optimize the test suite size in regression testing," *Csi Transactions on Ict*, vol. 4, no. 2-4, pp. 73–78, 2016.

[22] C. T. Lin, K. W. Tang, J. S. Wang, and G. M. Kapfhammer, "Empirically evaluating greedy-based test suite reduction methods at different levels of test suite complexity," *Science of Computer Programming*, vol. 150, pp. 1–25, 2017.

[23] C. Coviello, S. Romano, G. Scanniello, A. Marchetto, and A. Corazza, "Clustering support for inadequate test suite reduction," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018.

[24] F. Liu, J. Zhang, and E. Z. Zhu, "Test-suite reduction based on k-medoids clustering algorithm," in *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2017.

[25] A. Schuler, "Application of search-based software engineering methodologies for test suite optimization and evolution in mission critical mobile application development," in *11th Joint Meeting on Foundations of Software Engineering*, 2017.

[26] V. Garousi, R. Ozkan, and A. Betin-Can, "Multi-objective regression test selection in practice: An empirical study in the defense software industry," *Information Software Technology*, vol. 103, no. NOV., pp. 40–54, 2018.
[27] A. Panichella, F. Kifetew, and P. Tonella, "Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets," *IEEE Transactions on Software Engineering*, pp. 1–1, 2017.
[28] Q. Shi, J. Wan, Y. Feng, C. Fang, and Z. Chen, "Deepgini: Prioritizing massive tests to reduce labeling cost," *arXiv preprint arXiv:1903.00661*, 2019.
[29] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Input prioritization for testing neural networks," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2019, pp. 63–70.
[30] L. Zhang, X. Sun, Y. Li, and Z. Zhang, "A noise-sensitivity-analysis-based test prioritization technique for deep neural networks," *arXiv preprint arXiv:1901.00054*, 2019.
[31] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting operational dnn testing efficiency through conditioning," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 499–509.
[32] W. Tian, M. Lauer, and L. Chen, "Online multi-object tracking using joint domain information in traffic scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 1, pp. 374–384, 2019.
[33] L. M. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science*, vol. 259, no. 1-2, pp. 1–61, 2001.
[34] S. Gerasimou, H. F. Eniser, A. Sen, and A. Cakan, "Importance-driven deep learning system testing," *arXiv preprint arXiv:2002.03433*, 2020.

Sihan Xu received the BSc and Ph.D. degrees in computer science from Nankai University, Tianjin, China, in 2013 and 2018, respectively. She spent a year with National University of Singapore for her research. She is currently a research fellow in the College of Cyber Science, Nankai University. Her research interests include deep learning testing, software engineering, and AI security.

Zhiyu Wang is a graduate student in College of Cyber Science, Nankai University, China. He received his BEng degrees in computer science from Nankai University, Tianjin, China. His research focuses on deep learning testing and AI security.

Lingling Fan is an Associate Professor in College of Cyber Science, Nankai University, China. She received her Ph.D and BEng degrees in computer science from East China Normal University, Shanghai, China in June 2019 and June 2014, respectively. In 2017, she joined Nanyang Technological University (NTU), Singapore as a Research Assistant and then had been as a Research Fellow of NTU since 2019. Her research focuses on software security, program analysis and testing, and Android application analysis and testing. She got an ACM SIGSOFT Distinguished Paper Award at ICSE 2018. More information is available on https://lingling-fan.github.io/

Xiangrui Cai received his Ph.D. degree in computer science from Nankai University, Tianjin, China. He is currently an Assistant Professor with the College of Cyber Science, Nankai University. He has published several papers in international leading conferences and journals. His research interests include deep learning, natural language processing, time series analysis, and healthcare data mining.

Hua Ji received his Ph.D. degree in computer science from Nanjing University, Nanjing, China. He is currently a Distinguished Professor with the College of Cyber Science, Nankai University, Tianjin, China. His research interests include trusted AI, deep learning testing, and distributed systems.

Siau-Cheng Khoo received the Ph.D. degree in computer science from Yale University in 1992. He is currently an Associate Professor with School of Computing, National University of Singapore, Singapore. His research interests include specification mining, code analytics, static and dynamic program analysis, domain-specific languages, and aspect-oriented programming.

Brij B. Gupta received the PhD degree from IIT Roorkee, India. He was a postdoctoral research fellow in University of New Brunswick, Fredericton, Canada. He is currently working as an assistant professor with the Department of Computer Engineering, National Institute of Technology Kurukshetra, India. He spent more than six months with the University of Saskatchewan, Saskatoon, Canada, to complete a portion of his research. He has visited several countries to present his research. His biography is selected to publish in the 30th Edition of prestigious Marquis Who's Who in the World (2012). He has published more than 45 research papers in international journals and conferences of high repute. His research interest includes information security, cyber security, cloud computing, web security, intrusion detection, computer networks, and phishing. He is member of ACM, SIGCOMM, the Society of Digital Information and Wireless Communications (SDIWC), Internet Society, the Institute of Nanotechnology, and a Life member of the International Association of Engineers and the International Association of Computer Science and Information Technology. He has also served as a technical program committee member of more than 20 international conferences worldwide. In 2009, he was selected for Canadian Common-wealth Scholarship and awarded by the Government of Canada Award($10 000). He is an editor of various international journals and magazines.