Natural Adversaries: Fuzzing Autonomous Vehicles with Realistic Roadside Object Placements

Yang Sun Haoyu Wang Christopher M. Poskitt Jun Sun
School of Computing and Information Systems, *Singapore Management University*, Singapore
{yangsun.2020, haoyu.wang.2024}@phdcs.smu.edu.sg, {cposkitt, junsun}@smu.edu.sg

***Abstract*—The emergence of Autonomous Vehicles (AVs) has spurred research into testing the resilience of their perception systems, i.e., ensuring that they are not susceptible to critical misjudgements. It is important that these systems are tested not only with respect to other vehicles on the road, but also with respect to objects placed on the roadside. Trash bins, billboards, and greenery are examples of such objects, typically positioned according to guidelines developed for the human visual system, which may not align perfectly with the needs of AVs. Existing tests, however, usually focus on adversarial objects with conspicuous shapes or patches, which are ultimately unrealistic due to their unnatural appearance and reliance on white-box knowledge. In this work, we introduce a black-box attack on AV perception systems that creates realistic adversarial scenarios (i.e., satisfying road design guidelines) by manipulating the positions of common roadside objects and without resorting to "unnatural" adversarial patches. In particular, we propose TRASHFUZZ, a fuzzing algorithm that finds scenarios in which the placement of these objects leads to substantial AV misperceptions—such as mistaking a traffic light's colour—with the overall goal of causing traffic-law violations. To ensure realism, these scenarios must satisfy several rules encoding regulatory guidelines governing the placement of objects on public streets. We implemented and evaluated these attacks on the Apollo autonomous driving system, finding that TRASHFUZZ induced violations of 15 out of 24 traffic laws.**

***Index Terms*—Autonomous vehicles; perception systems; fuzz testing; adversarial scenarios; safety-critical systems.**

I. INTRODUCTION

Autonomous Vehicles (AVs) are rapidly advancing, with several Level-4 systems successfully deployed in real traffic environments [1]–[4]. This progress has intensified interest in evaluating the resilience of their perception systems, which must reliably interpret not only vehicles and road obstacles but also roadside objects. Everyday items such as trash bins, billboards, and greenery are placed according to guidelines [5]–[10] designed for human drivers, and may not align with the requirements of AV perception.

Most existing adversarial tests focus on conspicuous patches or irregularly shaped objects [11]–[16], such as MSF-ADV's visibly altered traffic cone [11]. These attacks typically pursue objectives such as making an object "ignored" or mislocalised and often require white-box access to perception models. While such misperceptions may expose safety issues, their unrealistic appearance and impractical assumptions limit applicability on real roads.

These limitations motivate two questions. First, can AV perception be deceived by natural, guideline-compliant roadside objects—those appearing entirely benign to human observers?

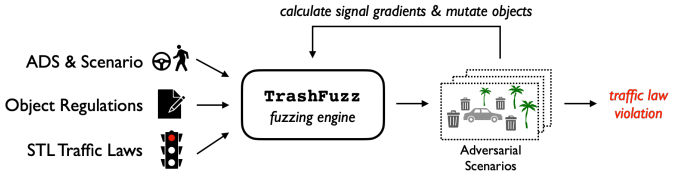

Fig. 1: TRASHFUZZ workflow: mutating roadside-object placements to find natural scenarios that induce an autonomous vehicle to violate traffic laws

Second, if such misperceptions occur, can they escalate into serious safety issues, such as collisions or violations of traffic laws [17]? Demonstrating these rare but plausible scenarios in simulation is essential before AVs encounter them in the wild.

To address these questions, we introduce TRASHFUZZ, a black-box fuzzing approach that systematically searches for guideline-compliant object placements capable of misleading an Autonomous Driving System (ADS) into unsafe behaviour. TRASHFUZZ generates scenarios using naturally shaped, patch-free objects (e.g., bins, benches, trees, hydrants), evaluates them in a high-fidelity simulator, and uses a greedy search to move "closer" to violating Signal Temporal Logic (STL) specifications of traffic laws. The fuzzer is constrained by regulatory documents—including the Code of Practice for Works on Public Streets [5], [6], greenery-provision guidelines [7], and placement rules from major waste-management companies [8]–[10]—ensuring all scenarios remain realistic. Figure 1 illustrates the workflow.

Unlike LawBreaker [17], which mutates the behaviour of agents (e.g., other vehicles or pedestrians) to induce traffic-law violations, TRASHFUZZ treats regulation-compliant roadside object layouts themselves as the adversarial input space. It introduces naturalness as a constraint, formalising real-world placement guidelines as executable rules that are enforced during scenario generation. Within this constrained space, TRASHFUZZ performs gradient-guided black-box optimisation over object types, positions, and orientations to minimise STL robustness and induce perception-driven law violations.

We implement TRASHFUZZ for the Baidu Apollo ADS [3] with the Unity-based LGSVL simulator [18], using the Unity Asset Store [19] to provide a wide range of everyday roadside objects. TRASHFUZZ automatically enforces placement constraints and searches for scenarios that trigger STL-defined law violations. In our evaluation, TRASHFUZZ efficiently gener-

ated scenarios that induced Apollo to violate 15 of 24 testable traffic laws. Most cases caused hesitation or failure to proceed; others induced aggressive behaviour, including one scenario in which an innocuous arrangement of bins overloaded the perception system and caused the ADS to misinterpret a red light as green.

These findings highlight that existing placement guidelines, designed primarily for human drivers, may be inadequate in the presence of AVs, and that testing pipelines must account for subtler, object-induced anomalies.

II. Background and Problem

In this section, we review the design of state-of-the-art ADS perception systems, define the attack goal and threat model, the DSLs for specifying safety properties, and highlight the key design challenges our solution must address.

A. ADS Perception Systems

State-of-the-art ADSs such as Apollo [20], Autoware [21], and Waymo [2] share two key perception characteristics: multi-sensor integration and the use of Deep Neural Networks (DNNs) for classification and segmentation.

Integration of Multiple Sensors. Modern Level-4 AVs typically fuse data from at least two sensor modalities—commonly cameras, LiDAR, and radar—to obtain a more accurate and robust view of the environment. While this multi-sensor setup enhances perception reliability, it does not guarantee resilience. For example, MSF-ADV [11] shows that carefully crafted objects can deceive both camera- and LiDAR-based detectors despite the presence of multiple sensors.

DNN-based Classification and Segmentation. DNNs are widely used in ADS perception pipelines for detecting and segmenting objects from camera and LiDAR data [20], [21]. However, DNNs are known to be vulnerable to adversarial manipulation. Prior work demonstrates that adversarial objects or patches can cause detectors to ignore obstacles [11], misinterpret roadside billboards [22]–[24], deviate lane-centering controllers [14], or corrupt depth estimation [15]. These examples are largely white-box attacks involving conspicuous or irregular adversarial patterns.

Defending DNNs against such attacks remains challenging. Although several defence mechanisms [25]–[28] show partial effectiveness, surveys [29], [30] emphasise that no universal solution exists. Thus, systematic testing of ADS perception systems continues to be essential.

B. Attack Goal and Threat Model

Attack Goal. We propose a black-box adversarial attack on the perception systems of ADSs, with the goal of inducing behaviour in which the AV violates traffic laws. This must be achieved by objects that follow conventional road design guidelines and appear benign to humans (i.e., no patches or unusual shapes). Such scenarios are termed "natural": they are subsets of all possible scenarios and are characterised by restrictions on the placement of objects like bins, billboards, and greenery. We formulate the attack goal as follows:

$$\phi \ ::= \mu \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \, \mathcal{U}_I \, \phi_2$$
$$\mu \ ::= f(x_0, x_1, \ldots, x_k) \ \sim \ 0, \quad \sim \in \{>, \geq, <, \leq, =, \neq\}$$

Fig. 2: Syntax of our specification language, a fragment of Signal Temporal Logic (STL). Here, ϕ denotes a formula, μ is an atomic predicate over real-valued functions f of variables x_i, and $I = [l, u]$ is a bounded or unbounded time interval

Definition 1 (Attack Goal)**.** *Let ϕ denote a user-specific specification of ADS behaviour, and φ denote a formula specifying the rules of natural scenarios. Then, our attack goal is to find a scenario λ such that $\lambda \vDash \varphi$ and there exists a trace π of the ADS within λ such that $\pi \nvDash \phi$.*

In this paper, the attack is divided into two tasks: (1) generate a scenario λ in which the placement of roadside objects satisfies some formalised regulatory rules φ; and (2) optimise the placement of objects to induce a misperception that causes the ADS to violate a traffic law specification ϕ.

Threat Model. Our attack assumes a black-box setting, in which the attacker has no access to the internal details of the perception system and its models. The attacker is only able to observe the outputs of the perception system and chassis information of the ADS. Furthermore, the attacker does not possess any knowledge of the inner architecture of the ADS or any algorithms that it utilises. This ensures that the attack can be used to test the resilience of any ADS by treating its perception system as a black box. The attacker's ability is limited to modifying and moving roadside objects. They are further *restricted to natural object placements* that satisfy government regulations [31] and company guidelines [8]–[10] ensuring scenarios are realistic. Furthermore, the attacker cannot utilise adversarial patches, as these appear as visually unnatural and require white-box knowledge of the perception models, which in our case the attacker does not have.

C. Specifying Safety Properties

Our attack goal is to cause misperceptions that result in the violation of safety properties. In the context of AVs, safety should not simply mean the absence of collisions, but also adherence to the rules of the road that drivers are supposed to abide by. To this end, we adopt the property specification language used by LawBreaker [17], as well as the project's existing specifications of the traffic laws of China and Singapore. The specification language is based on STL formulas, and is evaluated with respect to traces of scenes, providing a way to automatically determine whether a tester-defined property was violated or not in a simulated run of the ADS. We highlight the key features of the specification language below (the full syntax and semantics is given in [17]).

The high-level syntax of our specification language is formalised in Figure 2. Temporal formulas ϕ are built from atomic predicates μ using Boolean connectives and the bounded temporal operator $\mathcal{U}_I$ (interpreted as 'until over interval I'). A time interval I is of the form $[l, u]$, where l

and u are non-negative real-valued bounds. We adopt standard abbreviations: the *eventually* operator $\Diamond_I \phi$ is syntactic sugar for $true\ \mathcal{U}_I\ \phi$, and the *always* operator $\Box_I \phi$ abbreviates $\neg\Diamond_I \neg\phi$. When $I = [0, \infty]$, the interval may be omitted.

An atomic predicate μ is a relational constraint over a multivariate function $f(x_0, x_1, \ldots, x_k)$ and a constant threshold 0, using standard relational operators. These variables x_i are language-defined signals or features, such as vehicle speed, object distance, or perception confidence scores.

Example II.1. *Suppose we have a signal variable* $speed = \langle speed(0), speed(1), \ldots, speed(n)\rangle$*, which represents the autonomous vehicle's speed throughout its journey. Then, we can create a simple Boolean expression* $\mu = speed(t) < 100$ *to check whether the speed of the vehicle is larger than* $100km/h$*, as illustrated in Fig 2. Note that* μ *can be regarded as a proposition of the form* $100 - speed(t) > 0$ *or* $speed(t) - 100 < 0$*. To verify whether* μ *holds true at all time steps, we can straightforwardly incorporate the temporal logic symbol "always", resulting in the formulation of* $\varphi = \Box(speed < 100)$.

A specification is evaluated with respect to a trace π of *scenes*, denoted as $\pi = \langle \pi_0, \pi_1, \pi_2 \ldots, \pi_n\rangle$, where each scene π_i is a valuation of the propositions at time step i and π_0 reflects the state at the start of a simulation. The language follows the standard semantics of STL (see e.g., [32]).

D. Design Challenges

The goals above imply two key design challenges:

C1. Restricting to natural scenarios. A key objective of our work is to develop attack scenarios that appear as completely natural to humans. Previous work, such as NADE [33], has focused on replicating natural driving behaviours of vehicles. In contrast, our focus is on the strategic placement of typical roadside objects, including (but not limited to) trash bins, trees, and fire hydrants. A significant challenge is knowing when the placement of such objects crosses the boundary between something innocuous to something suspicious. To that end, we constrain the placement of objects according to existing regulations and guidelines (e.g., [8]–[10], [31]), which we use to standardise the notion of natural scenarios.

In addition, by restricting our scenarios to natural ones, we impose constraints that significantly reduce the available space of modifications, requiring testing algorithms that navigate them more creatively. We cannot arbitrarily modify the environment that the ADS is driving in. For example, placing a trash bin directly in the middle of the road should be forbidden in our testing context, as this is clearly an unnatural scenario that violates standard guidelines on bin placement.

C2. Large black-box search space. Despite restricting the placement of objects to conventional road design guidelines, there remains an enormous number of possible combinations of placements, making the search space vast and challenging to navigate, particularly in a black-box setting. For example, even when limiting to only 10 permissible placement locations, if each of them can be occupied by one of 10 different possible objects, we already have 10^{10} combinations of object placements. Hence, an effective search algorithm is necessary.

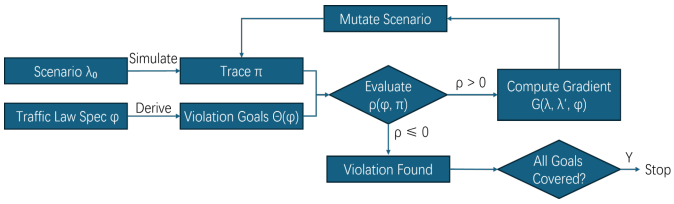

Fig. 3: Overview of the TRASHFUZZ approach

TABLE I: Specifications for natural scenarios, based on public road design guidelines

$rule1$	The fixed position attack objects should not be on the road or footway. $dis(obj.fix, road) \neq 0 \wedge dis(obj, footway) \neq 0$
$rule2$	The fixed position attack objects (OG boxes) should be at least 0.6 m away from the main road (2 lanes). $lanenum(road) \geq 2 \implies dis(obj.fix, road) > 0.6$
$rule3$	The fixed position attack objects cannot be near a minor road (1 lane). $lanenum(road) < 2 \implies dis(obj.fix, road) > 10$
$rule4$	The splay corner of the entrance culvert, bin centre access, substation access, MDF room access, and fire engine access must be at least 1 m from palms, 1.5 m from small/medium trees, and 2.5 m from large trees. $(tree.h < 1.5 \implies dis(tree, corners) > 1) \wedge (tree.h \geq 1.5 \wedge tree.h < 5 \implies dis(tree, corners) > 1.5) \wedge (tree.h \geq 5 \implies dis(tree, corners) > 2.5)$
$rule5$	A scupper pipe/drain must be at least 1 m from palms, 1.5 m from small-/medium trees, and 2.5 m from large trees. $(tree.h < 1.5 \implies dis(tree, pipe) > 1) \wedge (tree.h \geq 1.5 \wedge tree.h < 5 \implies dis(tree, pipe) > 1.5) \wedge (tree.h \geq 5 \implies dis(tree, pipe) > 2.5)$
$rule6$	Trees must be at least 3 m from a lamp post. $dis(tree, lamp) > 3$
$rule7$	Fixed objects (e.g., OG box, manhole, SCV box, lighting box, traffic light) must be at least 2 m from small/medium trees, and 2.5 m from large trees. $(tree.h < 5 \implies dis(tree, obj.fix) > 2) \wedge (tree.h \geq 5 \implies dis(tree, obj.fix) > 2.5)$
$rule8$	Trees must be at least 3 m from the edge of a footpath crossing. $dis(tree, footpath) > 3$
$rule9$	Trash bins should not block roads or footpaths. $dis(bin, road) > 0 \wedge (dis(bin, footpath) > 0 \vee footpath.width - bin.width > 1.5)$
$rule10$	Trash bins must be at least 0.5 m from each other and from mailboxes, walls, lamp posts, utility boxes, vehicles, etc. $dis(bin, obj) \geq 0.5$
$rule11$	Wheels/handles of bins must be oriented away from the road. $bin.direction \times road.direction < 0.2$
$rule12$	Ensure no object lies between trash bins and the road. $dis(obj, road) + dis(bin, obj) > dis(bin, road)$

III. TRASHFUZZ METHODOLOGY

We propose TRASHFUZZ to address these challenges. Figure 3 provides an overview, which we elaborate on below.

A. Generating Natural Scenarios (C1)

As mentioned, we overcome C1 by utilising existing regulations and guidelines for the placement of various roadside objects. In particular, we refer to government documents including the Code of Practice for Works on Public Streets (COP) [5] and Guidelines on Greenery Provision and Tree Conservation (GGPTC) [7]. The COP, formulated by Singapore's Land Transport Authority, delineates the necessary procedures and regulations for conducting activities on public roads, whereas the GGPTC, introduced by Singapore's National Parks Board, governs the placement and preservation of trees within urban landscapes. For the placement of trash bins, we refer to the

guidelines provided by three prominent recycling companies: HDS, KIMBLE, and GRANGER [8]–[10].

We formalise the notion of natural scenario by translating the regulations/guidelines described in these documents into mathematical formulas. Though in natural language, the guidelines are quite specific, allowing for a relatively direct translation of the rules to formulas. Table I lists these 12 specifications that jointly define natural scenarios, all of which must be satisfied. Specifications 1–3 refer to section 6.5 of the COP [5]. Specifications 4–8 refer to pages 95-97 of the GGPTC [7]. Specifications 9–11 pertain to key aspects of the guidelines provided by the waste recycling companies [8]–[10]. Note that while we have focused on guidelines from one country to ensure consistency between our rules, the general approach of formalising rules can be applied to the guidelines of any other country or jurisdiction.

In our rules, the function $dis(a, b)$ returns the distance between two objects, a and b, while the function $lanenum(road)$ returns the count of lanes within the designated $road$. Each distinct category of objects is denoted by a class variable. For example, the class $tree$ serves as a general representation for trees and includes a characteristic attribute, $tree.h$, corresponding to height.

It is crucial to have a diverse range of attack objects at our disposal. Several simulators, including AWSIM [34] and LGSVL [18], are built upon the Unity Real-Time Development Platform [35]. The Unity Asset Store [19] has a vast library of 3D asset packages, with an open-door policy that allows anyone to contribute their creations. Each asset package provides necessary elements, such as meshes of 3D objects and 2D textures, for constructing objects inside the simulators. We can access thousands of objects that are commonly encountered during driving such as rocks, trash bins, and hydrants for free. Hence, our strategy involves exploring Unity's asset library [19] to discover suitable adversarial objects.

In this work, we leverage a variety of freely accessible 3D asset packages from the Unity Asset Store [19] to integrate commonly encountered roadside objects. These include trash bins in a variety of shapes and colours, benches constructed from different materials and exhibiting various shapes, a wide array of tree types, hydrants, and trash bags. Crucially, these objects can be effortlessly spawned within Unity-based simulators, such as LGSVL [18]. We provide all these assets on our website [36], offering a convenient way for users to integrate them into the LGSVL simulator environment directly.

B. Greedy Gradient-Guided Black-Box Search (C2)

We require an efficient algorithm for identifying attack scenarios in a black-box setting. Given our objective is to induce ADS behaviours that violate traffic law specifications, it is important that the algorithm can uncover *different* ways to violate the specification. For instance, consider the specification $(speed > 10) \wedge (speed < 100)$. There are two distinct ways to violate it: first by inducing $speed$ to a value smaller than 10, and second by inducing it to a value larger than 100.

TABLE II: An example of an encoded scenario

Adversarial Objects	forward	right	rotation	type
`obstacle1`	25.04	2.63	25.78°	Bench1
`obstacle2`	49.45	6.23	9.21°	BigTrashBin
`obstacle3`	60.55	4.14	91.98°	TrashBin(Yellow)
`obstacle4`	57.85	7.98	49.89°	TrashBin(Red)
`obstacle5`	18.95	-9.83	290.48°	TrashBin(Green)

In this section, we present the detailed design of our algorithm. Given the time-consuming nature of simulator-based evaluations for ADSs, our primary goal is to attain convergence as fast as possible. To achieve this, our algorithm is *greedy*, in the sense of initially exploring adversarial scenarios that appear most likely to violate the given specification, and is also *gradient guided*, in the sense that it systematically expands its exploration based on these initial findings.

We present the algorithm in three parts. First, we delve into the encoding of (natural) scenarios as vectors, which forms the foundation of subsequent steps (e.g., mutation). Second, we present the design of our objective function which provides a quantified way to sufficiently cover different ways of violating a specification, ensuring diversity in the results. Finally, we introduce our new greedy gradient-guided search algorithm, which forms the core of our approach.

1) Encoding of Scenarios: We represent each adversarial scenario as a vector ϕ with dimensions $n \times d$. Here, n refers to the number of adversarial objects, while d corresponds to the four essential dimensions of each adversarial object: *forward*, *right*, *rotation*, and *type*. The *forward* and *right* dimensions indicate the relative position of the adversarial object with respect to the start position of the ego vehicle. For example, if the value for *forward* is 10, and the value for *right* is -2, the object is located at the left front of the ego vehicle. The *rotation* dimension represents the placement angle of the adversarial object. The *type* is a sequence number corresponding to the adversarial objects library, allowing us to easily identify and categorise different types of adversarial objects. In addition, each scenario also provides the initial and destination positions for the ADS journey.

Table II provides an example scenario encoding. Within this scenario, there are five adversarial objects. Four of these objects, namely a bench, a large trash bin, a yellow trash bin, and a red trash bin, are strategically placed to the front-right of the ego vehicle's starting point, each with distinct rotation angles (designated as $obstacle1 \ldots 4$). Additionally, a green trash bin is positioned in the front-left with a rotation angle of 290.48 degrees, labelled as $obstacle5$.

2) Coverage-based Objective Function Design: We treat the attack generation process as an optimisation problem.

Definition 2. *Let Θ be a set of traffic law specifications and Λ a set of adversarial scenarios. Our optimisation problem is:*

$$Minimise : \sum_{\phi \in \Theta} \min_{\lambda \in \Lambda} \{\rho(\phi, \pi_\lambda)\}$$

where π_λ is the trace of ADS in the face of an adversarial scenario λ and function $\rho(\phi, \pi)$ is the robustness value of trace π evaluated under specification ϕ.

We thus require a set of adversarial scenarios that minimises the robustness function across several property specifications. Intuitively, the robustness function reflects the *distance* to a specification violation, meaning it measures how close the current trace π is to violating the specification. First, we must establish a precise definition for the fundamental *objective function*, denoted as $\rho(\phi, \pi)$. Second, we require a systematic approach to replace the specification with a set of smaller formulas (Θ), each one characterising a different way to violate the original specification.

We define our objective function using the quantitative semantics of STL [32], [37], [38], which produces a numerical *robustness* degree.

Definition 3 (Quantitative semantics)**.** *Given a trace π and a formula ϕ, the* quantitative semantics *is defined as the robustness degree $\rho(\phi, \pi, t)$, computed as follows. Recall that propositions μ are of the form $f(x_0, x_1, \cdots, x_k) \sim 0$.*

$$\rho(\mu, \pi, t) = \begin{cases} -\pi_t(f(x_0, x_1, \cdots, x_k)) & \text{if } \sim \text{ is } \leq \text{ or } < \\ \pi_t(f(x_0, x_1, \cdots, x_k)) & \text{if } \sim \text{ is } \geq \text{ or } > \\ \mid \pi_t(f(x_0, x_1, \cdots, x_k)) \mid & \text{if } \sim \text{ is } \neq \\ - \mid \pi_t(f(x_0, x_1, \cdots, x_k)) \mid & \text{if } \sim \text{ is } = \end{cases}$$

where t is the time step and $\pi_t(e)$ is the valuation of expression e at time t in π;

$$\begin{aligned} \rho(\neg\phi, \pi, t) &= -\rho(\phi, \pi, t) \\ \rho(\phi_1 \wedge \phi_2, \pi, t) &= \min\{\rho(\phi_1, \pi, t), \rho(\phi_2, \pi, t)\} \\ \rho(\phi_1 \vee \phi_2, \pi, t) &= \max\{\rho(\phi_1, \pi, t), \rho(\phi_2, \pi, t)\} \\ \rho(\phi_1 \ \mathtt{U}_\mathtt{I} \ \phi_2, \pi, t) &= \sup_{t_1 \in t+\mathtt{I}} \min\{\rho(\phi_2, \pi, t_1), \inf_{t_2 \in [t, t_1]} \rho(\phi_1, \pi, t_2)\} \end{aligned}$$

where $t + I$ is the interval $[l + t, u + t]$ given $I = [l, u]$. □

Intuitively, this quantitative semantics computes how 'close' the ADS is from violating the given property specification ϕ. Note that the smaller $\rho(\phi, \pi, t)$ is, the closer π is to violating ϕ. If $\rho(\phi, \pi, t) \leq 0$, this means ϕ is violated. We write $\rho(\phi, \pi)$ to denote $\rho(\phi, \pi, 0)$; $\pi \vDash \phi$ to denote $\rho(\phi, \pi, t) > 0$; and $\pi \nvDash \phi$ to denote $\rho(\phi, \pi, t) \leq 0$. Note that time is discrete in our setting.

Example III.1. *Let $\varphi = \square(speed < 100)$, i.e., the vehicle's speed must always remain below 100 km/h. Suppose π is $\langle(speed \mapsto 0, \dots), (speed \mapsto 0.5, \dots), \cdots (speed \mapsto 90, \dots)\rangle$, where the ego vehicle's max speed is 90km/h at the last time step. We have $\rho(\varphi, \pi) = \rho(\varphi, \pi, 0) = min_{t \in [0, |\pi|]}(100 - \pi_t(speed)) = 10$. This means that trace π satisfies φ, and the robustness value is 10.*

Next, we transform a specification into a set of smaller formulas, such that the violation of any one formula implies the *violation of the original specification*. The rationale is that this will allow us to evaluate the robustness degree with respect to multiple different ways of violating the original specification, rather than just focusing on the 'easiest' one. Here, we adopt the methodology of LawBreaker [17], which defines a method to derive a set of violation goals based on the given complex specification ϕ. We write $\Theta(\phi)$ to denote a set of 'violation goals' of ϕ, i.e., that satisfy the following proposition:

Proposition III.2. *For STL formulas ϕ and traces π:*

$$\forall \varphi \in \Theta(\phi).\ \pi \nvDash \varphi \implies \pi \nvDash \phi$$

The set of violation goals $\Theta(\phi)$ is calculated as follows:

$$\begin{aligned} &\Theta(\mu) = \{\mu\}; \Theta(\neg\phi) = \Theta(N(\phi)); \\ &\Theta(\phi_1 \wedge \phi_2) = \Theta(\phi_1) \cup \Theta(\phi_2); \Theta(\bigcirc\phi) = \{\bigcirc x \mid x \in \Theta(\phi)\}; \\ &\Theta(\phi_1 \ \mathcal{U}_\mathcal{I} \ \phi_2) = \{x \ \mathcal{U}_\mathcal{I} \ y \mid x \in \Theta(\phi_1) \wedge y \in \Theta(\phi_2)\}; \end{aligned}$$

Here, μ is a Boolean expression and function $N(\phi)$ returns the negation of the formula ϕ without operator $\neg$. This eliminates the need for an additional calculation to determine $\Theta(\neg\phi)$, as all formulas containing the operator $\neg$ have already been transformed into their equivalent formulas without operator $\neg$.

Proposition III.3. *For STL formulas ϕ and traces π:*

$$\forall \phi.\ \forall \pi.\ \rho(N(\phi), \pi) = \rho(\neg\phi, \pi).$$

Definition 4 (Specification Negation)**.** *Given a specification formula ϕ, the negation of this formula, $N(\phi)$, is defined:*

$$\begin{aligned} &N(\mu) = n(\mu); N(\neg\phi) = \phi; N(\phi_1 \wedge \phi_2) = N(\phi_1) \vee N(\phi_2); \\ &N(\phi_1 \vee \phi_2) = N(\phi_1) \wedge N(\phi_2); N(\bigcirc\phi) = \bigcirc N(\phi); \\ &N(\phi_1 \ \mathtt{U}_\mathtt{I} \ \phi_2) = (\square_\mathtt{I} N(\phi_2)) \vee (N(\phi_2) \ \mathtt{U}_\mathtt{I} \ (N(\phi_1) \wedge N(\phi_2))); \end{aligned}$$

where $n(\mu)$ is defined as follows:

$$n(\mu) = \begin{cases} e_1 \geq (resp.\ >) e_2 & \text{if } \mu \text{ is } e_1 < (resp.\ \leq) e_2; \\ e_1 \leq (resp.\ <) e_2 & \text{if } \mu \text{ is } e_1 > (resp.\ \geq) e_2 \\ e_1 \neq (resp.\ =) e_2 & \text{if } \mu \text{ is } e_1 = (resp.\ \neq) e_2 \end{cases}$$

Propositions III.2 and III.3 can be proved by structural induction. We refer the readers to [17] for the detailed proofs.

Example III.4. *The value of $\Theta(\phi)$ is calculated recursively. For example, given a specification $\phi = \square(\mu_1 \wedge \mu_2)$, we can obtain $\Theta(\phi)$ as follows:*

1) *First, we apply Θ to the primitive elements: $\Theta(\mu_1) = \{\mu_1\}, \Theta(\mu_2) = \{\mu_2\}$*
2) *Then we have: $\Theta(\mu_1 \wedge \mu_2) = \Theta(\mu_1) \cup \Theta(\mu_2) = \{\mu_1, \mu_2\}$*
3) *Given $\Theta(\mu_1 \wedge \mu_2)$, we can get: $\Theta(\square(\mu_1 \wedge \mu_2)) = \{\square(\mu_1), \square(\mu_2)\}$*
4) *The final result is $\Theta(\phi) = \{\square(\mu_1), \square(\mu_2)\}$.*

i.e., there exist two violation goals for ϕ, and the violation of either one of them can result in the violation of ϕ.

3) Greedy Gradient Guided Query: With the derivation of violation goals reviewed, we now propose our novel greedy gradient-guided query algorithm. We first describe the gradient calculation process before presenting the algorithm as a whole.

In our black-box setting, gradient calculations are important for guiding the generation of adversarial scenarios. These gradients are constructed by differentiating between the original scenario and the scenario resulting from modifications. We define the gradient calculation below:

Algorithm 1: The TRASHFUZZ Algorithm

Input: ϕ (traffic law), n (number of adversarial objects), and M (maximal queries)
Output: A test suite Γ

Let $\Theta_r = \Theta(\phi)$ be the set of uncovered formulas in $\Theta(\phi)$;
Let $seed$ be a pair storing a gradient value and mutable element; initially the gradient value $seed.gradient$ is 0 and the mutable element $seed.element$ is $Null$;
Let λ_0 be a randomly generated adversarial scenario;
Let Γ be an empty set;
while Θ_r *is not empty and not timeout* **do**
    Execute λ_0 via simulation and obtain trace π_0;
    Mutate each adversarial object in λ_0 and get a few adversarial scenarios $\lambda_1, \ldots, \lambda_n$;
    Execute $\lambda_1, \ldots, \lambda_n$ via simulation and obtain trace $\pi_1, \ldots, \pi_n$;
    for *each* π_k *in* $\pi_1, \ldots, \pi_n$ **do**
        for *each* φ *in* Θ_r **do**
            if $\rho(\varphi, \pi_k) \leq 0$ **then**
                Remove φ from Θ_r; Add s_k into Γ;
            end
            else if $|G(\lambda_0, \lambda_k, \varphi)| > |seed.gradient|$ **then**
                $seed.gradient = G(\lambda_0, \lambda_k, \varphi)$;
                Store the mutated element of s_k to $seed.element$;
            end
        end
    end
    Generate the next s_0 based on $seed$;
end
return Γ

Definition 5. *Let ϕ be a property specification, λ be the initial scenario, λ' be the scenario after modification, and $E(\lambda)$/$E(\lambda')$ denote the encoding of scenario λ/λ'. The gradient calculation is formally defined as follows:*

$$G(\lambda, \lambda', \phi) = \frac{\rho(\phi, \pi) - \rho(\phi, \pi')}{E(\lambda) - E(\lambda')}$$

where π and π' are the ADS traces under scenarios λ and λ', respectively.

Intuitively, the gradient calculation reveals the extent to which modifying the scenario influences the behaviour of the ADS. A positive gradient value signifies that the current modification enhances robustness with respect to the property specification, while a negative gradient value suggests a decrease in robustness due to the modification. Furthermore, the magnitude of the gradient value reflects the magnitude of the modification's impact on the robustness value.

The overall TRASHFUZZ algorithm is shown in Algorithm 1. First, we generate a random scenario as the initial test case and decompose the traffic law specification ϕ to obtain Θ_r, then initialise $seed$ with $seed.gradient$ value 0 and $seed.element$ value $Null$. Note that $seed$ is a pair storing a gradient value and the corresponding mutable element.

Next, using the initial adversarial scenario λ_0, we execute it to generate the trace π_0. The adversarial scenario contains several adversarial objects, and for each of these objects, we apply random mutations to create new scenarios. Note that the mutations have been designed to ensure that the ADS remains compliant with the rules listed in Table I, as well as with other physical constraints such as the requirement for objects to be grounded. For each newly generated scenario, we also execute it to produce a trace and assess it against all the remaining specifications in set Θ_r. We remove φ from the set if it is violated, i.e., if $\rho(\varphi, \pi_i) \leq 0$, and add the corresponding scenario λ_i to the output set Γ. After executing and processing all the cases generated by the initial test case, we utilise the gradient calculation defined above to choose the seed for the next round. In essence, we designate the case with the largest absolute gradient value as the seed for the subsequent round.

This process is repeated until all the elements in set $\Theta(\phi)$ are covered or the maximal number of generations M (as defined by the user) has been reached.

IV. IMPLEMENTATION AND EVALUATION

In this section, we present our implementation of TRASHFUZZ for the Apollo ADS and evaluate its effectiveness in identifying natural scenarios that lead to traffic law violations, as well as other key aspects. For additional experimental materials and results, we direct readers to our website [36].

A. Implementation

Implementing TRASHFUZZ involves three broad steps. First, some basic integration with and between the existing ADS and simulator, including a communication bridge, a trace generator, and the means of both checking a specification as well as calculating its robustness with respect to a trace. Second, the generation of natural scenarios consisting of (benign-appearing) adversarial roadside objects, as detailed in Section III-A. Finally, the implementation of our greedy gradient-based fuzzing engine, as elaborated on in Section III-B.

The target of our evaluation is the LGSVL [18] simulator with version 7.0 of the Apollo ADS [3] (the latest version stable with LGSVL). To establish communication between Apollo and LGSVL, we utilise the official communication bridge, CyberRT [39]. For trace generation and specification evaluation/robustness, we use the implementation strategy described by LawBreaker [17]. In particular, the trace generator creates a trace π by processing messages subscribed to by the ADS, whereas for specifying traffic laws, we utilise the domain-specific language provided by LawBreaker. To implement the quantitative semantics of those traffic laws, we embedded the *RTAMT* tool [38] to compute the robustness of the specifications with respect to the trace obtained from the bridge. (Note that implementing TRASHFUZZ for Autoware/-CARLA would follow a largely similar strategy, except for substituting the ROS bridge [40] for CyberRT.)

In order to generate natural scenarios, we sourced common roadside objects from the Unity Asset Store [19]. As LGSVL is a Unity-based simulator, these could be directly placed into scenarios. The roadside objects we utilised in TRASHFUZZ have been assembled into a single library that we have made available online [36]. To ensure the objects are placed according to conventional road design guidelines, we constrained their placement according to the functions in Table I. These functions take a scenario as input and return a Boolean value

TABLE III: Percentage of valid scenarios

Atk Obj		1	2	3	4	5	6	7
Rand	rule1	79.9	63.7	50.1	39.9	32.5	25.2	20.3
	rule2-3	77.7	60.2	46.2	35.2	28.3	21.1	16.4
	rule4-6	91.1	83.5	76.3	69.0	63.6	57.4	53.2
	rule9	85.1	73.7	62.9	53.9	45.8	38.3	34.1
	rule10	100	99.99	99.98	99.99	99.97	99.96	99.95
	rule11	89.2	80.7	71.5	64.9	58.0	51.8	45.8
	rule12	100	100	100	100	100	100	100
	all	**56.0**	**32.0**	**17.6**	**10.1**	**5.8**	**3.1**	**1.6**
Ours	**all**	**100**	**100**	**100**	**100**	**100**	**100**	**100**

to indicate whether the generated scenario is valid or not according to the guidelines.

Finally, we integrated the above components and implemented our greedy gradient-based fuzzing engine as described in Section III-B. Our experiments were conducted using two machines, each equipped with 32GB of memory, an Intel i7-10700k CPU, and an RTX 2080Ti graphics card. These two machines run on Linux (Ubuntu 20.04.5 LTS) and Windows (Windows 10 Pro) operating systems, respectively.

B. Evaluation

We conducted experiments to answer four Research Questions (RQs):

- **RQ1**: Can TRASHFUZZ produce valid scenarios?
- **RQ2**: Can natural attack scenarios cause traffic law violations?
- **RQ3**: Can our greedy gradient-based search algorithm cover more violations than the most comparable fuzzer?
- **RQ4**: Do other ADS perception stacks exhibit similar object-induced misclassifications?

RQ1: Can* TRASHFUZZ *produce valid scenarios? Given the specifications shown in Table I, we first investigate whether TRASHFUZZ can effectively generate valid scenarios that satisfy them. We compare TRASHFUZZ with randomly generated scenarios. Our evaluation involved analyzing 10,000 randomly generated scenarios, each with a fixed number of attack objects. Note that in these scenarios, we randomly place attack objects within a designated area, i.e., a 150m $\times$ 60m space in front of the AV. The results are shown in Table III. Here, the numbers (1 to 7) in the first row indicate the number of attack objects. We present the percentage of scenarios that satisfy the given specification. A higher percentage value indicates a greater number of valid scenarios. Rule2 and rule3 have been merged into one specification concerning the distance of fixed-position objects from the road. Likewise, rule4–6 are combined into one specification that evaluates the positioning of trees. This analysis demonstrates that a majority of the randomly generated scenarios are invalid with respect to public road design guidelines. Moreover, as the number of attack objects increases, the likelihood of scenarios deviating from these guidelines rises. This result indicates that most randomly generated scenarios are not natural. For instance, these objects are often placed directly on the road, a violation of `rule1` or `rule9` as outlined in Table I. As shown by the final row of Table III, TRASHFUZZ generates scenarios that satisfy the placement rules by construction (invalid mutations are discarded via lightweight constraint checks), yielding a much higher proportion of natural scenarios than random placement.

TABLE IV: Violations of traffic laws; note that TRASHFUZZ operates under natural scenario restrictions

Traffic Laws		LawBreaker	TRASHFUZZ	Law Concerns
Law38	sub1	$\surd$	$\surd$	green light
	sub2	$\surd$	$\surd$	yellow light
	sub3	$\surd$	$\surd$	red light
Law44		$\surd$	$\surd$	lane change
Law45	sub1	$\times$	$\times$	speed limit
	sub2	$\times$	$\times$	speed limit
Law46	sub2	$\surd$	$\surd$	speed limit
	sub3	$\surd$	$\surd$	speed limit
Law47		$\surd$	$\surd$	overtake
Law50		$\times$	$\times$	reverse
Law51	sub3	$\surd$	$\surd$	traffic light
	sub4	$\surd$	$\surd$	traffic light
	sub5	$\surd$	$\surd$	traffic light
	sub6	$\times$	$\times$	traffic light
	sub7	$\times$	$\surd$	traffic light
Law52 sub2-4		$\times$	$\times$	priority
Law53		$\times$	$\times$	traffic jam
Law57	sub1	$\surd$	$\surd$	left turn signal
	sub2	$\surd$	$\surd$	right turn signal
Law58		$\surd$	$\surd$	warning signal
Law59		$\surd$	$\surd$	signals
Law62	sub8	$\times$	$\times$	honk

RQ2: Can natural attack scenarios cause traffic law violations? To answer this question, we applied our fuzzing algorithm to systematically test all testable traffic law specifications (including the multiple sub-laws) provided with the LawBreaker [17] tool. Our experiment covered all 'testable' laws, excluding those that cannot be tested due to limitations in the simulator. For instance, traffic regulations related to traffic lights with arrow lights are non-testable because LGSVL only supports traffic lights with circular lights.

The results are summarised in Table IV. Here, we present all the testable laws applicable to AVs within the platform. However, certain other relevant laws cannot be tested due to limitations in the available maps. For example, law 49 specifies that vehicles should not make U-turns at railway crossings, sharp bends, steep slopes, or tunnels. Unfortunately, the current maps of existing platforms (LGSVL and CARLA) do not include representations of these specific locations, rendering them untestable under the current conditions.

Within this table, the 'LawBreaker' and 'TRASHFUZZ' columns indicate whether Apollo violated the specific traffic regulation listed in the first two columns. We mark $\surd$ for a traffic law ϕ if and only if $\pi \not\models \phi$ happened for a scenario generated by LawBreaker or TRASHFUZZ. We repeated each test case at least three times to ensure reproducibility and mitigate potential flakiness in simulation-based ADS testing [41]. Note that the tools are violating laws using different mutation strategies: LawBreaker mutates the driving behaviour of other vehicles and pedestrians, whereas TRASHFUZZ mutates only the placement of roadside objects subject to the roadside design guidelines (Table I). TRASHFUZZ's strategy is thus

much more challenging due to the additional restrictions.

As can be seen from the table, even while constrained to satisfy road design guidelines, TRASHFUZZ still manages to incite violations of a majority of the traffic laws. In comparison to LawBreaker, TRASHFUZZ covers all the traffic laws that LawBreaker violates. Additionally, TRASHFUZZ has the capability to activate an additional sub-law, specifically sub-law 7 of Article 51, which regulates that turning vehicles yield to straight-moving vehicles, whereas pedestrians and right-turning vehicles travelling in the opposite direction yield to left-turning vehicles. The activation of this supplementary sub-law is triggered by the presence of adversarial objects along the roadside. Note that TRASHFUZZ found an additional law violation compared to LawBreaker *despite the restriction* to modifying roadside object placements only.

We identified two broad categories of issues among the tests generated by TRASHFUZZ, which we elaborate on below. These misjudgments stem from flaws in the perception system of the ADS. While the camera images and lidar points received by the ADS from the simulator appear normal, the system arrives at incorrect judgments. Note that we repeat the following issues at least three times to ensure their reproducibility.

Classification and Segmentation Errors. The first category concerns misclassification of adversarial objects or failures in the segmentation algorithm. For instance, trash bins may be misclassified as pedestrians or bicycles, causing the ADS to treat them as moving agents rather than stationary obstacles (e.g., waiting for a 'pedestrian' that is actually a bin). Placing two bins in close proximity can also cause the perception system to interpret them as a single object such as a vehicle, potentially confusing downstream decision-making and causing hesitation or inappropriate behaviour at intersections. Similarly, when a bin lies adjacent to a bench, the ADS may fail to segment them separately and instead recognise a single unknown object. These errors persist across multiple frames, propagating through the system and inducing misbehaviour such as hesitation or failure to proceed.

Overloaded Perception System. The second category of issues concerns the perception system of the ADS, which in general, is complex and consists of multiple algorithms and modules. In issues of this category, a few adversarial objects are strategically placed at specific positions along the ADS's route to deny the service of the traffic light perception system of the ADS. In this situation, the traffic light perception system becomes overloaded, resulting in anomalous outputs. We observed two possible consequences of this situation in our tests. In the first, the roadside objects deceive the AV into perceiving the traffic light ahead as a red or yellow traffic light. This would cause the vehicle to interpret the intersection as a signal to stop indefinitely, regardless of the absence of an actual red traffic light or any other vehicles or pedestrians. Such a misperception can disrupt the normal flow of traffic and potentially lead to congestion or delays. In the second, we deceive the autonomous vehicle into perceiving the traffic light ahead as a green traffic light. Consequently, the detection system always indicates a green signal for the traffic light ahead, regardless of its actual state. This situation can lead to dangerous driving behaviours, such as rushing red lights, endangering the AV itself and others on the road.

TABLE V: Number of traffic law violation goals (out of a possible 82) covered for the Apollo ADS

Num	Driver	Alg.	R1	R2	R3	R4	Avg
7	Apollo	LawBreaker	23	15	19	16	**18.25**
		TRASHFUZZ	17	22	19	21	**19.75**
6	Apollo	LawBreaker	15	15	13	22	**16.25**
		TRASHFUZZ	19	16	18	21	**18.5**
5	Apollo	LawBreaker	13	16	14	13	**14**
		TRASHFUZZ	21	16	15	20	**18**
4	Apollo	LawBreaker	17	16	13	14	**15**
		TRASHFUZZ	17	15	16	14	**15.5**
3	Apollo	LawBreaker	14	9	14	11	**12**
		TRASHFUZZ	17	16	16	14	**15.75**
2	Apollo	LawBreaker	11	14	11	10	**11.5**
		TRASHFUZZ	14	15	10	13	**13**
1	Apollo	LawBreaker	12	14	10	9	**11.25**
		TRASHFUZZ	13	11	17	12	**13.25**
0	Apollo	-	7	9	5	7	**7**

RQ3: Can our greedy gradient-based search algorithm cover more violations than the most comparable fuzzer? To answer this question, we do a baseline comparison of TRASHFUZZ's gradient-based search algorithm against the coverage-based fuzzing algorithm of LawBreaker [17], which is based on a genetic algorithm (GA) [42]. In particular, after decomposing traffic law specifications into violation goals (Section III-B2), we compare the coverage of the two tools in the sense of how many of those goals they are able to induce Apollo to satisfy.

LawBreaker's GA—the baseline in this experiment—begins by creating a generation of test scenarios and subsequently applies mutations and crossovers to produce the next generation. Intuitively, individuals with superior performance effectively survive and contribute to the creation of the subsequent generations. In LawBreaker's fuzzing algorithm, we initialise the population with 30 individuals and run the algorithm for 20 generations, resulting in a total of 620 test cases generated in a single execution. For TRASHFUZZ, we configure it to allow a maximum of 620 queries, aligning with LawBreaker's fuzzing algorithm. Note that LawBreaker concentrates on altering the behavior of NPC vehicles and pedestrians on the road, while TRASHFUZZ focuses on modifying the placement of roadside objects. Therefore, to ensure a fair and meaningful comparison, we adopt identical test case encoding methodologies for both LawBreaker and TRASHFUZZ, as detailed in Section III-B1. Additionally, we apply the same scenario restrictions outlined in Table I to LawBreaker's fuzzing algorithm, to ensure our comparison solely assesses efficiency of the algorithms under equivalent conditions.

The results are summarised in Table V. The overall traffic law specification ϕ we use is derived from LawBreaker [17], which contains 24 different Chinese traffic laws that can be decomposed into 82 possible violation goals ($\Theta(\phi)$). In the table, the 'Num' column denotes the quantity of adversarial objects involved in the respective attack scenario, ranging from

TABLE VI: Testing Apollo's perception system

Object	Type	Vehicle?	Bicycle?	Ped?	Ignore?
TrashBin(Grey)	Movable	√	√	√	×
TrashBin(Yellow)	Movable	√	√	√	×
TrashBin(Blue)	Movable	√	√	√	×
TrashBin(Red)	Movable	√	√	√	×
BigTrashBin	Movable	√	×	×	×
ShoppingCart	Movable	×	×	×	√
WarningStand	Movable	×	×	×	√
TrashBag	Movable	×	×	×	√
Bench0	Fixed-Pos	×	√	×	√
Bench1	Fixed-Pos	√	√	×	√
BusStopPole	Fixed-Pos	×	×	×	√
Hydrant	Fixed-Pos	×	×	×	√
Tree0	Fixed-Pos	×	×	×	√
Tree1	Fixed-Pos	×	×	×	√
Tree2	Fixed-Pos	×	×	×	√

TABLE VII: Testing Autoware's perception system

Object	Type	Vehicle?	Bicycle?	Ped?	Ignore?
Chair	Movable	×	√	√	√
Parasol	Movable	×	√	×	√
TrashBin	Movable	×	×	√	×
LandscapeSign	Fixed-Pos	√	×	×	√
Mailbox	Fixed-Pos	×	×	√	√
Bench	Fixed-Pos	√	×	×	√
Pine	Fixed-Pos	×	×	√	√
Stall	Fixed-Pos	√	×	×	√
Fence	Fixed-Pos	√	×	×	√
Hydrant	Fixed-Pos	×	×	√	√

0 to 7. When the number is 0, it indicates that scenario is a common one without any attack. The 'R1–4' columns indicate how many of the 82 possible violation goals for ϕ are covered across four rounds of experimentation, where the average number is given in the last column. As shown, for both LawBreaker and TRASHFUZZ, the coverage of violations increases proportionally with the number of adversarial objects. Furthermore, TRASHFUZZ surpasses LawBreaker in covering more violations within a limited number of queries.

RQ4: Do other ADS perception stacks exhibit similar object-induced misclassifications? The attack carried out by TRASHFUZZ exploits vulnerabilities in ADS perception algorithms, which are not limited to Apollo. Fully integrating TRASHFUZZ with other ADS stacks is non-trivial due to architectural differences (e.g., Autoware's ROS-based microservice architecture). Therefore, rather than running the full TRASHFUZZ optimisation loop, we conduct a preliminary perception-level probe using Autoware with the CARLA simulator [43]. We place roadside objects at random locations along the AV's path, vary their orientations, and evaluate whether the perception system ignores or misclassifies them.

In Table VI and Table VII, we present the results of the evaluations for Apollo (v7) and Autoware (Autoware.ai v1.14), respectively. The 'Object' column displays the name of each object, while the 'Type' column indicates whether the object is movable or of fixed-position. The 'Vehicle?', 'Bicycle?', 'Pedestrian?', and 'Ignore?' columns respectively check whether the perception of the ADS classifies the object as a vehicle, bicycle, pedestrian, or if it is ignored. It is worth noting that both Apollo and Autoware exhibit misclassifications for common objects along the road under certain circumstances. These misclassifications expose potential vulnerabilities in the perception system of these autonomous driving systems.

Although our evaluation focuses on Apollo, we conducted these preliminary experiments with Autoware to explore TRASHFUZZ's generalisability. These showed that adversarial layouts generated by TRASHFUZZ can also induce classification errors in Autoware, suggesting that the underlying attack strategies may transfer across different ADSs. However, full pipeline integration is currently limited by Autoware's ROS-based microservice architecture, which complicates the incorporation of our constraint-based scenario generation and STL-guided optimisation loop. Future work will therefore focus on adapting TRASHFUZZ for broader ADS compatibility by: (i) developing ROS-compatible modules for scenario evaluation and STL violation detection, (ii) introducing an abstraction layer between the TRASHFUZZ optimiser and modular perception outputs, and (iii) deploying the framework on additional platforms such as CARLA with Autoware.Universe. These steps would extend the applicability of TRASHFUZZ and enable robust cross-platform adversarial testing.

V. DISCUSSION

A. Threats to Validity

Simulation-based testing introduces validity concerns. ADSs require significant computational resources, and insufficient resources may add latency. To mitigate this, we ran Apollo and the simulator on separate machines connected via Ethernet, ensured memory cleanliness, and avoided background tasks. Under these conditions, Apollo's modules maintained an average latency below 0.1s. We also reproduced each identified issue at least three times. While environment-induced effects cannot be fully ruled out, the surfaced behaviours remain valuable for understanding and improving ADS robustness.

B. Practicality of TRASHFUZZ Attacks

We briefly discuss the practical assumptions required to realise attacks such as TRASHFUZZ in comparison with existing approaches. Table VIII summarises the prerequisites of several representative attacks in the literature.

Within the table, the categories 'Patch?', 'Shape?', 'Placement?', and 'Special?' indicate whether the attack requires a specialised adversarial patch, modifies the shape of an object, relies on carefully chosen placement of physical objects, or requires specialised equipment unlikely to be found on the roadside (e.g., commercial drones, digital LiDAR spoofers, or projectors). The columns 'White-B?', 'Grey-B?', and 'Black-B?' indicate the assumed level of access to the ADS. A checkmark (√) denotes that a particular prerequisite is required. For example, the MSF-Attack [11] modifies the shape of roadside objects (e.g., traffic cones) to make them invisible to camera and LiDAR detectors, and therefore requires both object shape manipulation and controlled placement.

As shown in Table VIII, TRASHFUZZ does not require adversarial patches or shape modifications to objects, avoiding the practical challenges associated with designing and deploying visually conspicuous artefacts. Instead, the attack operates by manipulating the placement of ordinary roadside objects

TABLE VIII: Prerequisites for applying different attacks

Method	Ours	MSF [11]	ADSDoS [44]	DeepBB [22]	DirtyRd [14]	CarPatch [15]	Frustum [12]	AttackZ [16]
Patch?	×	×	×	√	√	√	×	√
Shape?	×	√	×	×	×	×	×	×
Placement?	√	√	√	√	√	√	√	√
Special?	×	×	×	×	×	×	√	√
White-box?	×	√	×	√	√	√	×	√
Grey-box?	×	×	√	×	×	×	×	×
Black-box?	√	×	×	×	×	×	√	×

while respecting existing road-design guidelines. Furthermore, TRASHFUZZ operates entirely in a black-box setting, without requiring access to the internal models of the ADS. Some prior attacks share certain characteristics (e.g., placement-based manipulation), but may require additional assumptions such as knowledge of specific system components or specialised equipment. Overall, TRASHFUZZ relies only on common roadside objects and black-box access to the ADS, suggesting that such scenarios are plausible to realise in practice.

C. Limitations and Future Work

Recent AV stacks increasingly incorporate multimodal foundation models (e.g., GPT-4o [45], [46]) that jointly process vision, language, and sensor signals [47], [48]. Extending TRASHFUZZ to such systems requires moving beyond physical object placement to semantically grounded perturbations, e.g., misleading textual cues or scene-level affordance manipulations that bias multimodal decision-making. Language-guided scene optimisation is a promising direction for testing next-generation AVs.

Our evaluation focuses on object-placement-based scenario generation and does not include reinforcement-learning approaches (e.g., CRASH [49]), which offer a complementary direction. While our work centres on vulnerability discovery, natural extensions include exploring defences such as perception filtering or refining road-design rules (e.g., VisionGuard [50]). We also focus on static roadside objects; extending the framework to dynamic elements such as mobile billboards [24] would further broaden the evaluation.

VI. RELATED WORK

Attacks for AVs. ADSs are complex and security-critical systems, yet difficult to harden against adversarial behaviours. Existing work largely falls into two categories.

Adversarial-patch attacks. Several studies place adversarial patches on roadside billboards to mislead perception [22]–[24], or apply inconspicuous patches on the road surface to disrupt lane-centering [14]. Patches placed on leading vehicles can corrupt depth estimation [15], while projector-based "shadow" patterns induce segmentation errors [16]. These attacks require suspicious-looking artifacts and are typically white-box, limiting practical deployability.

Object-placement attacks. Approaches in this category manipulate the placement or shapes of roadside objects. MAF-ADV makes traffic cones invisible to camera and LiDAR, then positions them to trigger collisions [11]. Other attacks position reflective objects (e.g., cardboard, signs, drones) to inject false LiDAR points [13], [44]. The frustum attack exploits camera blind spots to mislead multi-sensor fusion [12]. Wan et al. [51] and Wang et al. [52] show that cones, lights, and other objects can induce overly conservative or incorrect behaviours.

These attacks generally rely on conspicuous or non-standard objects and assume white-box access. In contrast, TRASHFUZZ performs a black-box attack using only everyday objects placed in guideline-compliant ("natural") locations, yet still induces hesitation or aggressive driving in Apollo.

Critical Scenario Generation. Several approaches generate adversarial or safety-critical scenarios by perturbing background-vehicle behaviours. Genetic-algorithm fuzzers (e.g., AV-Fuzzer [53], DoppelTest [54]) and evolutionary searches such as AutoFuzz [55] optimise scenarios for collisions or off-road outcomes. NADE [33] perturbs naturalistic trajectories, while CRISCO [56] and MOSAT [57] assign impactful behaviours or solve multi-objective formulations for inducing failures. Reinforcement-learning approaches (e.g., DeepCollision [58]) similarly learn to steer scenes toward collisions.

Recent work has also explored objectives beyond inducing collisions [59]. LawBreaker [17] and ABLE [60] target traffic-law coverage; TARGET [61] uses LLMs to extract formalised specifications; other methods evaluate metamorphic consistency [62], derive metamorphic relations from traffic rules to generate transformed road images [63], maximise map-area coverage [64], or frame behavioural diversity as coupon-collector problems [65]. MORLOT [66] guides AVs to violate multiple safety metrics. Recent behaviour-centric methods such as Tian et al. [56] and BehAVExplor [67] extract influential interaction patterns or maximise behavioural diversity to elicit failures—approaches that complement our object-centric, regulation-constrained focus.

Overall, these scenario-generation methods generally mutate *agent behaviours*, overlooking the effects of naturally-placed *roadside objects* focused on by TRASHFUZZ.

VII. CONCLUSION

We introduced TRASHFUZZ, a black-box approach for testing ADS perception by generating natural, patch-free adversarial scenarios that comply with regulatory and company guidelines for roadside object placement. Using a greedy gradient-based fuzzing algorithm, TRASHFUZZ identifies placements of everyday objects that exploit perception vulnerabilities and trigger traffic-law violations.

We implemented TRASHFUZZ for the Apollo ADS with the LGSVL simulator and built a diverse library of realistic roadside objects. Our evaluation uncovered natural scenarios causing Apollo to violate 15 Chinese traffic laws, including a case where a benign bin layout overloaded perception and triggered a traffic-light misclassification.

These results expose robustness gaps in modern ADS perception and suggest that existing roadside-object guidelines may need revision—e.g., standardising the appearance or dimensions of common objects such as trash bins—to better support AV perception systems.

ACKNOWLEDGEMENT

This work is partially supported by DTC project DTC-IGC-08.

REFERENCES

[1] SAE On-Road Automated Vehicle Standards Committee, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," *SAE International: Warrendale, PA, USA*, 2021.

[2] Waymo. (2025) Waymo Driver. https://waymo.com/waymo-driver/. Online; accessed December 2025.

[3] Baidu. (2025) Apollo. https://www.apollo.auto/. Online; accessed December 2025.

[4] TuSimple. (2024) Autonomous driving technology designed for trucks. https://www.tusimple.com/technology/. Online; accessed September 2024.

[5] Land Transport Authority. (2025) Code of practice for works on public streets 1. https://www.lta.gov.sg/content/dam/ltagov/industry_innovations/industry_matters/development_construction_resources/Street_Work_Proposals/codes_of_practice/COP_for_Works_on_Public_Streets_Sep2018Ed.pdf. Online; accessed December 2025.

[6] Singapore Government. (2025) Street works (works on public streets) regulations. https://sso.agc.gov.sg/SL/SWA1995-RG2. Online; accessed December 2025.

[7] National Parks Board. (2023) Guidelines on greenery provision and tree conservation for developments. https://www.nparks.gov.sg/-/media/nparks-real-content/partner-us/nparks-handbook_version-4.pdf. Online; accessed December 2025.

[8] Homewood Disposal Service. (2025) How to place your cart at the curb. https://mydisposal.com/how-to-place-your-garbage-cart-at-the-curb. Online; accessed December 2025.

[9] Kimble Companies. (2024) Trash cart placement. https://www.kimblecompanies.com/trash-cart-placement. Online; accessed September 2024.

[10] Granger Waste Services. (2024) Cart placement. https://www.grangerwasteservices.com/wp-content/uploads/2021/09/Cart-Placement-Flyer-no-time.pdf. Online; accessed September 2024.

[11] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. A. Chen, M. Liu, and B. Li, "Invisible for both camera and LiDAR: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks," in *SP*. IEEE, 2021, pp. 176–194.

[12] R. S. Hallyburton, Y. Liu, Y. Cao, Z. M. Mao, and M. Pajic, "Security analysis of camera-LiDAR fusion against black-box attacks on autonomous vehicles," in *USENIX Security Symposium*. USENIX Association, 2022, pp. 1903–1920.

[13] Y. Zhu, C. Miao, T. Zheng, F. Hajiaghajani, L. Su, and C. Qiao, "Can we use arbitrary objects to attack LiDAR perception in autonomous driving?" in *CCS*. ACM, 2021, pp. 1945–1960.

[14] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, "Dirty road can attack: Security of deep learning based automated lane centering under physical-world attack," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 3309–3326.

[15] Z. Cheng, J. Liang, H. Choi, G. Tao, Z. Cao, D. Liu, and X. Zhang, "Physical attack on monocular depth estimation with optimal adversarial patches," in *ECCV (38)*, ser. LNCS, vol. 13698. Springer, 2022, pp. 514–532.

[16] R. Muller, Y. Man, Z. B. Celik, M. Li, and R. M. Gerdes, "Physical hijacking attacks against object trackers," in *CCS*. ACM, 2022, pp. 2309–2322.

[17] Y. Sun, C. M. Poskitt, J. Sun, Y. Chen, and Z. Yang, "LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles," in *ASE*. ACM, 2022, pp. 62:1–62:12.

[18] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Mozeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, "LGSVL simulator: A high fidelity simulator for autonomous driving," in *ITSC*. IEEE, 2020, pp. 1–6.

[19] Unity. (2025) Unity store. https://assetstore.unity.com/. Online; accessed December 2025.

[20] Baidu. (2025) Latest Apollo. https://github.com/ApolloAuto/apollo. Online; accessed December 2025.

[21] Autoware.AI. (2025) Autoware.AI. https://autoware.org/. Online; accessed December 2025.

[22] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu, "DeepBillboard: systematic physical-world testing of autonomous driving systems," in *ICSE*. ACM, 2020, pp. 347–358.

[23] N. Patel, P. Krishnamurthy, S. Garg, and F. Khorrami, "Adaptive adversarial videos on roadside billboards: Dynamically modifying trajectories of autonomous vehicles," in *IROS*. IEEE, 2019, pp. 5916–5921.

[24] ——, "Overriding autonomous driving systems using adaptive adversarial billboards," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11 386–11 396, 2022.

[25] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *ICLR (Poster)*. OpenReview.net, 2017.

[26] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015.

[27] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2016, pp. 582–597.

[28] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting classifiers against adversarial attacks using generative models," in *ICLR (Poster)*. OpenReview.net, 2018.

[29] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *CoRR*, vol. abs/1810.00069, 2018.

[30] ——, "A survey on adversarial attacks and defences," *CAAI Trans. Intell. Technol.*, vol. 6, no. 1, pp. 25–45, 2021.

[31] Singapore Government. (2025) Street works act 1995. https://sso.agc.gov.sg/Act/SWA1995#top. Online; accessed December 2025.

[32] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *FORMATS/FTRTFT*, ser. LNCS, vol. 3253. Springer, 2004, pp. 152–166.

[33] S. Feng, X. Yan, H. Sun, Y. Feng, and H. X. Liu, "Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment," *Nature Communications*, vol. 12, no. 1, p. 748, 2021.

[34] Autoware. (2025) AWSIM. https://tier4.github.io/AWSIM/. Online; accessed December 2025.

[35] Unity Real-Time Development Platform. (2025) Unity. https://unity.com/. Online; accessed December 2025.

[36] Y. Sun. (2025) TrashFuzz. Online; accessed December 2025. [Online]. Available: https://trashfuzz.github.io/

[37] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods Syst. Des.*, vol. 51, no. 1, pp. 5–30, 2017.

[38] D. Nickovic and T. Yamaguchi, "RTAMT: online robustness monitors from STL," in *ATVA*, ser. LNCS, vol. 12302. Springer, 2020, pp. 564–571.

[39] LGSVL. (2021) Cyberrt. https://github.com/lgsvl/CyberRT. Online; accessed December 2025.

[40] Carla. (2019) ROS bridge for Autoware and Carla. https://github.com/carla-simulator/carla-autoware. Online; accessed December 2025.

[41] O. Osikowicz, P. McMinn, and D. Shin, "Empirically evaluating flaky tests for autonomous driving systems in simulated environments," in *FTW@ICSE*. IEEE, 2025, pp. 13–20.

[42] S. Mirjalili, *Evolutionary Algorithms and Neural Networks - Theory and Applications*, ser. Studies in Computational Intelligence. Springer, 2019, vol. 780.

[43] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," in *CoRL*, ser. Proceedings of Machine Learning Research, vol. 78. PMLR, 2017, pp. 1–16.

[44] Y. Zhu, C. Miao, F. Hajiaghajani, M. Huai, L. Su, and C. Qiao, "Adversarial attacks against LiDAR semantic segmentation in autonomous driving," in *SenSys*. ACM, 2021, pp. 329–342.

[45] OpenAI, "GPT-4o system card," *CoRR*, vol. abs/2410.21276, 2024.

[46] ——, "Images and vision," https://platform.openai.com/docs/guides/images-vision, 2025, accessed December 2025.

[47] Z. Xu, Y. Zhang, E. Xie, Z. Zhao, Y. Guo, K. K. Wong, Z. Li, and H. Zhao, "DriveGPT4: Interpretable end-to-end autonomous driving via large language model," *IEEE Robotics Autom. Lett.*, vol. 9, no. 10, pp. 8186–8193, 2024.

[48] C. Kelly, L. Hu, B. Yang, Y. Tian, D. Yang, C. Yang, Z. Huang, Z. Li, J. Hu, and Y. Zou, "VisionGPT: Vision-language understanding agent using generalized multimodal framework," *CoRR*, vol. abs/2403.09027, 2024.

[49] A. Kulkarni, S. Zhang, and M. Behl, "CRASH: challenging reinforcement-learning based adversarial scenarios for safety hardening," *CoRR*, vol. abs/2411.16996, 2024.

[50] X. Han, H. Wang, K. Zhao, G. Deng, Y. Xu, H. Liu, H. Qiu, and T. Zhang, "VisionGuard: Secure and robust visual perception of autonomous vehicles in practice," in *CCS*. ACM, 2024, pp. 1864–1878.

[51] Z. Wan, J. Shen, J. Chuang, X. Xia, J. Garcia, J. Ma, and Q. A. Chen, "Too afraid to drive: Systematic discovery of semantic DoS vulnerability in autonomous driving planning under physical-world attacks," in *NDSS*. The Internet Society, 2022.

[52] W. Wang, Y. Yao, X. Liu, X. Li, P. Hao, and T. Zhu, "I can see the light: Attacks on autonomous vehicles using invisible lights," in *CCS*. ACM, 2021, pp. 1930–1944.

[53] G. Li, Y. Li, S. Jha, T. Tsai, M. B. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. K. Iyer, "AV-FUZZER: finding safety violations in autonomous driving systems," in *ISSRE*. IEEE, 2020, pp. 25–36.

[54] Y. Huai, Y. Chen, S. Almanee, T. Ngo, X. Liao, Z. Wan, Q. A. Chen, and J. Garcia, "Doppelgänger test generation for revealing bugs in autonomous driving software," in *ICSE*. IEEE, 2023, pp. 2591–2603.

[55] Z. Zhong, G. E. Kaiser, and B. Ray, "Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1860–1875, 2023.

[56] H. Tian, G. Wu, J. Yan, Y. Jiang, J. Wei, W. Chen, S. Li, and D. Ye, "Generating critical test scenarios for autonomous driving systems via influential behavior patterns," in *ASE*. ACM, 2022, pp. 46:1–46:12.

[57] H. Tian, Y. Jiang, G. Wu, J. Yan, J. Wei, W. Chen, S. Li, and D. Ye, "MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm," in *ESEC/SIGSOFT FSE*. ACM, 2022, pp. 94–106.

[58] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali, "Learning configurations of operating environment of autonomous vehicles to maximize their collisions," *IEEE Trans. Software Eng.*, vol. 49, no. 1, pp. 384–402, 2023.

[59] Y. Zhou, Y. Sun, Y. Tang, Y. Chen, J. Sun, C. M. Poskitt, Y. Liu, and Z. Yang, "Specification-based autonomous driving system testing," *IEEE Trans. Software Eng.*, vol. 49, no. 6, pp. 3391–3410, 2023.

[60] X. Zhang, W. Zhao, Y. Sun, J. Sun, Y. Shen, X. Dong, and Z. Yang, "Testing automated driving systems by breaking many laws efficiently," in *ISSTA*. ACM, 2023, pp. 942–953.

[61] Y. Deng, Z. Tu, J. Yao, M. Zhang, T. Zhang, and J. X. Zheng, "TARGET: traffic rule-based test generation for autonomous driving via validated LLM-guided knowledge extraction," *IEEE Trans. Software Eng.*, vol. 51, no. 7, pp. 1950–1968, 2025.

[62] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Commun. ACM*, vol. 62, no. 3, pp. 61–67, 2019.

[63] Y. Deng, J. X. Zheng, T. Zhang, H. Liu, G. Lou, M. Kim, and T. Y. Chen, "A declarative metamorphic testing framework for autonomous driving," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1964–1982, 2023.

[64] Z. Hu, S. Guo, Z. Zhong, and K. Li, "Coverage-based scene fuzzing for virtual autonomous driving testing," *CoRR*, vol. abs/2106.00873, 2021.

[65] F. Hauer, T. Schmidt, B. Holzmüller, and A. Pretschner, "Did we test all scenarios for automated and autonomous driving systems?" in *ITSC*. IEEE, 2019, pp. 2950–2955.

[66] F. U. Haq, D. Shin, and L. C. Briand, "Many-objective reinforcement learning for online testing of DNN-enabled systems," in *ICSE*. IEEE, 2023, pp. 1814–1826.

[67] M. Cheng, Y. Zhou, and X. Xie, "BehAVExplor: Behavior diversity guided testing for autonomous driving systems," in *ISSTA*. ACM, 2023, pp. 488–500.