Semantic Image Fuzzing of AI Perception Systems

Trey Woodlief
University of Virginia
Charlottesville, Virginia, USA
adw8dm@virginia.edu

Sebastian Elbaum
University of Virginia
Charlottesville, Virginia, USA
selbaum@virginia.edu

Kevin Sullivan
University of Virginia
Charlottesville, Virginia, USA
sullivan@virginia.edu

ABSTRACT

Perception systems enable autonomous systems to interpret raw sensor readings of the physical world. Testing of perception systems aims to reveal misinterpretations that could cause system failures. Current testing methods, however, are inadequate. The cost of human interpretation and annotation of real-world input data is high, so manual test suites tend to be small. The simulation-reality gap reduces the validity of test results based on simulated worlds. And methods for synthesizing test inputs do not provide corresponding expected interpretations. To address these limitations, we developed *semSensFuzz*, a new approach to fuzz testing of perception systems based on semantic mutation of test cases that pair real-world sensor readings with their ground-truth interpretations. We implemented our approach to assess its feasibility and potential to improve software testing for perception systems. We used it to generate 150,000 semantically mutated image inputs for five state-of-the-art perception systems. We found that it synthesized tests with novel and subjectively realistic image inputs, and that it discovered inputs that revealed significant inconsistencies between the specified and computed interpretations. We also found that it produced such test cases at a cost that was very low compared to that of manual semantic annotation of real-world images.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → *Perception*; Vision for robotics; • **Computer systems organization** → **Embedded and cyber-physical systems**.

KEYWORDS

semantic fuzzing, autonomous systems, perception

ACM Reference Format:
Trey Woodlief, Sebastian Elbaum, and Kevin Sullivan. 2022. Semantic Image Fuzzing of AI Perception Systems. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510212

1 INTRODUCTION

The perception-implementing layers of software in autonomous systems (ASs) are responsible for mapping raw sensor inputs to *semantic interpretations* that can inform decisions and actions in the physical world. Figure 1 illustrates a simplified AS pipeline where the world is sensed through a camera. The resulting image is then processed by *Percep* to generate an interpretation that the AS will use to inform navigation and other control decisions.

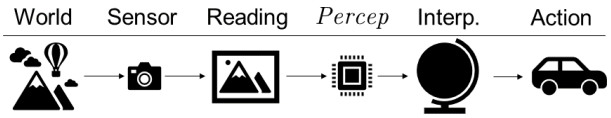


Figure 1: Autonomous System Perception Pipeline

Faulty perception systems can misinterpret the physical world, leading to dangerous, even deadly, actions. For example, Tesla has recently admitted that its ASs sometimes misinterpret parked cars, and the US National Highway Traffic Safety Administration has now opened an investigation into almost a dozen instances where a Tesla vehicle has crashed into a parked emergency vehicle with its lights flashing. Those mishaps have resulted in at least one fatality and multiple serious injuries [3, 35]. Other examples include miscalculating the existence or location of a vehicle, or failing to detect people in the planned trajectory of a vehicle [1, 12, 18, 36].

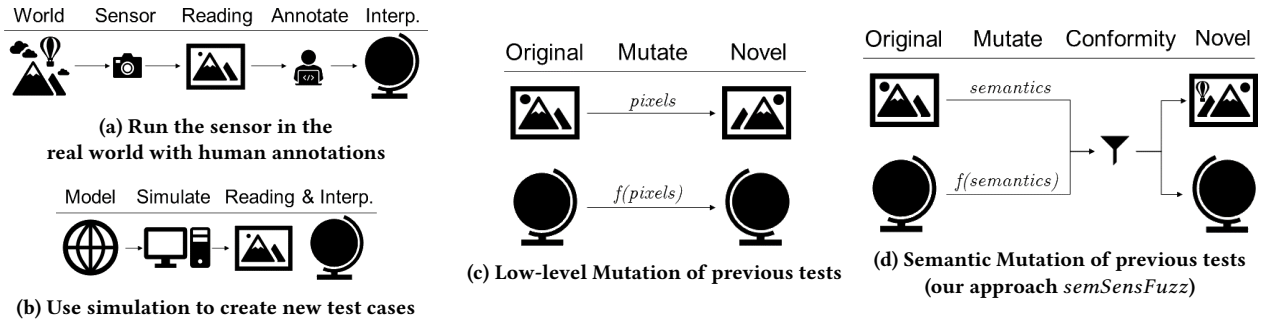
Such mishaps point to fundamental shortcomings in current methods of testing AS perception software. A key problem is that these machine-learned components tend to be trained using data from normal driving, with few if any opportunities to learn from rare but safety-critical events. When such events do occur, perception systems *must* accurately interpret the physical world.

The rarity of safety-critical but infrequent events then entails that real-world test driving will never be adequate from a testing perspective. A study by the RAND Corporation found that autonomous vehicles would have to drive hundreds of millions of miles to demonstrate reliability, and that doing so would take existing fleets tens or hundreds of years [16]. Along similar lines, Waymo Corporation safety validation methods convey opportunities to scale up testing using simulation, but ultimately conclude that reliance on real-world road driving for validation is required [21]. Augmentation of data captured in the real world has emerged as a potentially viable alternative to reduce the need for real-world driving [10, 26, 30, 37]. However, these techniques operate at the pixel-level, missing an opportunity to truly explore images that are semantically interesting in challenging the perception pipeline.

What this work proposes is an **approach to testing perception systems using sensor-reading/expected-interpretation test case pairs derived from real-world AS perception test cases (for which we already have ground truth interpretations), by mutating both sensor readings (e.g. images) and their expected interpretations in a coordinated manner**. Our long-term aim is to generate test cases that focus on rare, safety-critical inputs that include, for example, vehicles that are crashed, crossing into oncoming traffic, overturned, etc.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05.
https://doi.org/10.1145/3510003.3510212



(a) Run the sensor in the real world with human annotations

(b) Use simulation to create new test cases

(c) Low-level Mutation of previous tests

(d) Semantic Mutation of previous tests (our approach *semSensFuzz*)

Figure 2: Overview of test case generation approaches for perception layers of autonomous systems

This paper takes the first step towards our long-term vision by proposing and evaluating an approach for non-guided test generation that can, for example, incorporate vehicles and people into images. The approach leverages prior test cases as raw materials, mutating existing test inputs to automatically produce realistic and novel synthetic sensor data, and deriving interpretations by also mutating their existing interpretations. Testing with our approach then involves comparing these predicted outputs with the actual interpretation outputs produced by a given *system under test (SUT)*.

Our contributions are as follows:

(1) *semSensFuzz*, a new concept in testing of perception software for safety-critical autonomous systems, based on semantic mutations applied to (sensor reading, ground-truth interpretation) test case pairs.
(2) *semImFuzz*, a demonstration system targeting camera-based autonomous vehicle perception.
(3) Experiments using our approach to test five state-of-the-art perception systems, with results showing that our approach can produce realistic inputs and overall test case pairs that reveal problematical perception errors in these systems.

2 MOTIVATION AND RELATED WORK

Basic definitions. A **test case**, $t = (r, interp)$ for the perception system $Percep$ of an AS is a set of input sensor readings, r, paired with a valid output interpretation, $interp$, for r. Figure 3a shows an example of a single sensor reading r for an autonomous vehicle perception system and below it, Figure 3f shows the corresponding ground-truth interpretation $interp$. The system's performance is then judged based on an oracle that compares $Percep(r)$ and $interp$.

Conformity. ASs operate in the physical world and thus a sensor reading r used in a test case t must conform to the constraints of the real world. If r could have been acquired from (and is thus sufficiently realistic with respect to) some configuration of the real world, w, we say that r **conforms** with w. This notion is important as a failing t with nonconforming r is likely a false-positive—a failure that would not occur in the real world.

Figure 2 outlines three common existing procedures to generate a suite of t and our approach *semSensFuzz*, which we now explore.

2.1 Testing in the Real World

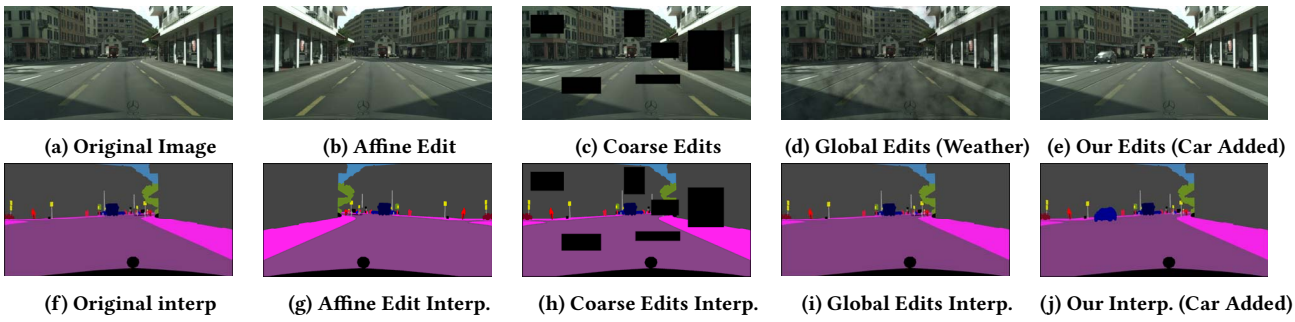
One common procedure, outlined in Figure 2a, is to operate the AS in the physical world while recording the sensor readings. This procedure is advantageous in that it can generate a large data set reflecting typical operating conditions. However, it is limited in that producing specific desired conditions (e.g., have a car suddenly turn in front of our fast moving vehicle) in the real world can be impractical, dangerous, and expensive [21]. Crafting the ground-truth interpretation also incurs great expense, requiring a human to manually annotate the sensor readings. In prior studies, producing high-quality annotations of camera images for a driving benchmark required on average over 90 minutes of human effort per image [7].

2.2 Testing in Simulation

Another common procedure, shown in Figure 2b, uses a model of the world embedded in a simulator to replace operating in the real world. Using the simulator's model of the world to automatically produce sensor readings and ground-truth interpretations enables constructing arbitrary simulated worlds at a lower cost [27, 31, 38]. However, this approach suffers from the simulation-reality fidelity gap as $w \neq w_{sim}$ [15]. This can diminish the value of the tests as the results may be simulation-specific [40], which is why they are often revalidated in the physical world [21]. That is, these tests may not conform to any configuration of the real world.

2.3 Generating Tests with Low-level Mutations

A third procedure mutates collected sensor readings to produce $(r, interp)$ and is illustrated in Figure 2c. Naive techniques follow in the vein of standard data augmentation techniques, performing global mutations such as affine transforms or adding noise [10, 26, 42]. For example, Figure 3b and its interpretation in Figure 3g show horizontally mirroring the image and its interpretation, while Figures 3c and 3h show the use of a mask that obscures parts of the image. More advanced strategies such as adding weather, shown in Figures 3d and 3i, are domain-specific but still globally applied across all pixels [30, 37, 42]. Mutation-based procedures can quickly generate many tests that, if the mutations are designed carefully, can also be conforming. That is much less likely, for example, for the box additions in 3c than for the fog addition in 3d. Furthermore, to automatically provide an oracle, this kind of procedure tends to not semantically affect the image. For example, adding fog should not affect the entities identified in the image—the interpretation of Figure 3i is the same as the original in Figure 3f. Additionally, recent work has examined ways to use mutation strategies to generate adversarial changes to the scene that do not change the semantics but are likely to cause changes in the AS's perception [2, 17, 43]. When these mutations do affect the image semantically, like in 3c,



(a) Original Image (b) Affine Edit (c) Coarse Edits (d) Global Edits (Weather) (e) Our Edits (Car Added)

(f) Original interp (g) Affine Edit Interp. (h) Coarse Edits Interp. (i) Global Edits Interp. (j) Our Interp. (Car Added)

Figure 3: Overview of image mutation techniques (best viewed on a screen)

the interpretation of those changes is unknown as shown in 3h where several people are cut off.

2.4 Generating Tests with Semantic Mutations

In this work we envision a more sophisticated kind of mutation that incorporates world semantics and is cognizant of what the readings mean (i.e., what the pixels in an image actually represent in the world). Such a mutation could, for example, add a car driving on the street as shown in Figure 3e with a corresponding interpretation in Figure 3j. We believe and later show how such semantic mutations have the potential to create test cases that are conforming but do not occur in existing test suites due to their uncommon occurrences in the physical world.

Figure 2d shows how our proposed approach, *semSensFuzz*, differs from prior mutation techniques in the semantic nature of the mutations. **Performing these mutations requires that mutated sensor readings continue to conform to the constraints of the real world**, leading to several challenges. On the interpretation side, we must craft domain-specific rules to validate potential mutations and determine what types of mutations are likely to yield perception failures, e.g. adding a pedestrian to a roadway. Furthermore, we must increase conformity likelihood by crafting preconditions based on the sensor modality. For example, for camera images, this may include not just that an entity is in a viable position, but also that the perspective, lighting, and shadows are conforming. Sensor reading mutations must be associated with interpretation mutations that will serve as an oracle. Section 3 describes how the approach tackles these challenges.

3 APPROACH

We describe our approach for using semantic mutations to test AS perception systems regardless of sensor modality. We formalize the problem definition, present the general approach, and instantiate the approach for camera-based perception systems for autonomous vehicles.

3.1 Problem definition

A **semantic mutation**, $\delta = (\delta_r, \delta_{interp})$, transforms a test case, $t = (r, interp)$, where r conforms with some configuration of the world, w, into a new test case, $t' = \delta(t) = (\delta_r(r), \delta_{interp}(interp)) = (r', interp')$, where r' also conforms with some world configuration, w', and $interp'$ is a valid interpretation of r'. For example, a mutation, δ^{addCar} may add a car to an image while mutating its interpretation to indicate that a car is now there.

We say that $Percep$ **passes** a mutated test case $\delta(t)$, if and only if $Percep(\delta_r(r)) = \delta_{interp}(interp)$, highlighting the symmetry of the mutation functions. Figure 4 illustrates this process. In practice, one might need to (and in our experiments we do) use approximate equality: $Percep(\delta_r(r)) - \delta_{interp}(interp) < \epsilon$ for a suitably small ϵ. At a conceptual level, our approach thus constitutes a form of metamorphic testing [32] where existing test cases are converted into new ones in which the correct outputs for transformed inputs are deduced using knowledge of properties of the physical world.

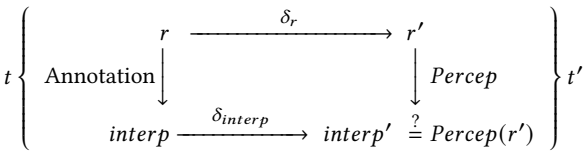


Figure 4: Semantic Mutation of Tests for Perception Systems

3.2 Semantic Mutations with *semSensFuzz*

A key insight driving *semSensFuzz* is that prior captured data sets can be leveraged as a source of initial test cases and of resources to design semantic mutations that are more likely to result in conforming tests. That insight guides the scope of mutations considered in any given domain, including means to ensure reasonable conformity of mutated inputs with possible real worlds.

First, we scope the space of all mutations Δ based on the entities that appear in the *interp*s of the available tests. For example, if the interpretations mark cars and people, we only allow mutations that add, remove, or change cars and people. We argue that since those are the criteria by which existing tests are judged, it is sensible to focus just on those to try to find unexpected $Percep$ behaviors.

Second, we associate a set of preconditions $Prec$ with each mutation δ, specified in terms of the *interp* that defines whether δ is applicable to a given t. If $Prec(interp)$ is not satisfied, then δ is not applicable to that test. For example, for a mutation that adds a car to an image, $Prec$ may be the existence of a road in *interp*; for changing the color of a car, $Prec$ would be the existence of at least one car in *interp*. The idea is to leverage an existing interpretation to determine mutation applicability. This is a key distinction

compared to other strategies that allows for our semantic mutations while respecting conformity. Instead of crafting mutations for sensor inputs and propagating the mutation to the interpretation without respect for changing semantics, we craft our mutations for the interpretation and then propagate to the sensor domain.

Third, we enable the parameterization of δ_r to employ real sensor data to increase the likelihood of conformity. For example, when adding a car to $t_i = (r_i, interp_i)$, if there exists a $t_j = (r_j, interp_j)$ that contains a car constrained by a similar context as per their $interp$s, then δ_r^{addCar} could put the car from r_j into r_i. In the case of images, we use the entity pose, perspective, lighting, and adjacent entities in the interpretation to define the context. Note that context is sensor modality and domain dependent, and that the richness of the resource set will affect the diversity of generated test cases. Additionally, we explore the integration of discriminators to determine conformity. We discuss this further in Section 4.4.4.

All of the mechanisms in our approach make a conscious trade off between conformity and a smaller space of available mutations.

3.3 Semantic Mutation of Images by *semImFuzz*

To further explore our approach, we now focus on the perception system of an autonomous vehicle that contains a single front-facing camera for sensing, and we concretize *semSensFuzz* to this specific sensor modality. We refer to this specific application as *semImFuzz*. As illustrated in Figure 3, for these systems the sensor input consists of a single camera image and the output consists of an interpretation of the world in the form of a per-pixel semantic annotation.

semImFuzz requires a data set of real-world camera data to build its resource set to serve as the basis of its mutations. Camera based perception systems are widely studied, with several available benchmarks [5, 7, 11] consisting of thousands of test cases of image and interpretation pairs. Each benchmark targets a different level of precision which affects the strength of the resource set.

We now explore the space of semantic mutations, Δ, for the domain of camera systems for autonomous vehicles. We first define the *semantic entity types* that can appear in images, e.g. car, bicyclist, etc. Each such entity has a state, e.g., orientation, color, position, lighting, etc. Finally we define a set of semantic actions: e.g., add entity, remove entity, change entity state. The space of mutations, Δ, is then the set of actions, each parameterized by an entity and a state, e.g. add (action) a car (entity) at location l (state).

In practice, semantically mutating images can be difficult due to the conformity requirement. For example, adding a car to a location in an image requires verifying that a car can exist in that location: e.g., the car cannot intersect another car. Removing an entity requires generating conforming image data with a known interpretation for the vacated space, a process known as semantic inpainting [20, 39]. Changing the pose of an entity requires rendering the entity in a conforming manner from a different perspective than it appeared in the original test case.

4 IMPLEMENTATION FOR THE AUTONOMOUS VEHICLE DOMAIN

To study the applicability of *semSensFuzz*, we created an extensible pipeline in Python for fuzzing perception systems of autonomous vehicles with a single front-facing camera, *semImFuzz*, that implements three mutations: changing the color of cars, adding cars, and adding pedestrians[1]. We begin with the system architecture and then discuss detailed implementations for the chosen mutations.

4.1 *semImFuzz* Architecture

Figure 5 outlines *semImFuzz*'s components and flow. Given a mutation δ, *semImFuzz* selects a test case t and queries the resource set for viable resources to perform δ. It then selects a resource and checks whether mutating using t, δ, and the selected resource will generate a conforming test. If so, it applies the mutation to generate t', otherwise it selects another resource and repeats.

semImFuzz requires a prior data set as a basis for mutation. In the autonomous vehicle domain, there are several such data sets, including KITTI [11], nuScenes [5], and Cityscapes [7]. Each provides $(r, interp)$ test cases where r is a camera image and $interp$ is a per-pixel annotation that provides a ground-truth interpretation based on the categories provided in the data set. As a preprocessing step, *semImFuzz* parses the prior test set to build a resource set of available data, e.g. a list of all cars and their poses.

Each data set has a different list of tracked categories with differing levels of precision and design choices for how to treat certain entities. For example, nuScenes has 7 labels for humans differentiating between, e.g. adults and children, while Cityscapes has 2 labels that distinguish between a pedestrian and a person riding a bike or motorcycle. We developed *semImFuzz* using the Cityscapes data set due to its popularity in related literature, the wide number of perception systems that have been developed to target the data set, and the availability of open sourced tools to evaluate performance. We note, however, that the design choices and available data in a data set influence the richness and precision of *semImFuzz*'s resource set and thus the mutations available. For example, when we aim to develop a mutation for adding a vehicle that has been involved in a crash to a test case, under the theory that a deformed vehicle would be more difficult to perceive, the data set's lack of this label will prevent *semImFuzz* from leveraging such a mutation.

We implemented each mutation using Numpy [13] along with OpenCV for Python [4] and Pillow [6], two common image processing libraries for Python. We created the framework in a modular fashion to easily incorporate additional mutations or data sets.

For our first implementation of *semImFuzz*, we chose to explore two mutations: changing the colors of entities within the scene and adding entities to the scene. We chose these mutations for initial study of *semImFuzz* due to our ability to perform those mutations using available image manipulation techniques. Although these mutations are relatively simple, we conjecture that if the simple mutations yield interesting test cases, then this will encourage further research to enable more advanced techniques which can then be incorporated. We describe the implementation details of each mutation in the following sections.

4.2 Changing Object Color

The simplest mutation we implemented changes the color of a single entity in the test case and was designed with the goal of changing

[1]Code and high-level algorithm descriptions are available at https://github.com/less-lab-uva/perception_fuzzing

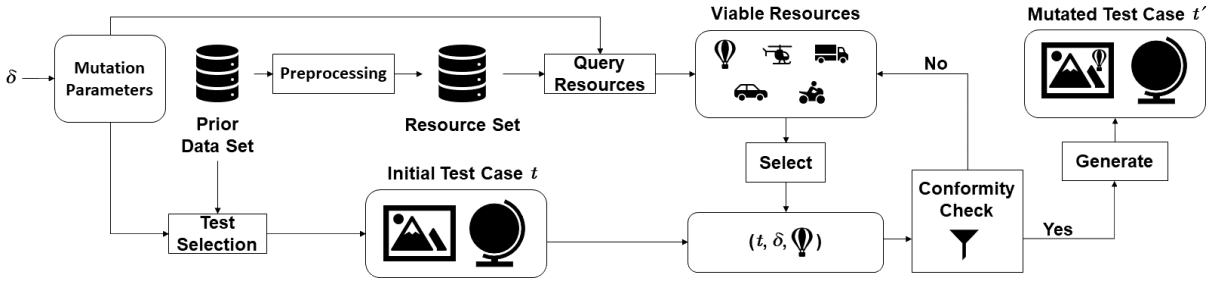


Figure 5: Pipeline for Semantic Mutation with *semImFuzz*

(a) Original Car

(b) Mutated Car

Figure 6: Applying color change mutation (best viewed on a screen)

(a) Original Car

(b) Mutated Car

Figure 7: Nonconforming color change mutation (best viewed on a screen)

the color of vehicles to mirror the physical parallel of repainting a vehicle. In the context of the SUTs we are examining, the color of the vehicle should not change the target perception category, and so $interp' = interp$. Thus, we examine how to produce r' from r.

4.2.1 Implementation. For this mutation, the test selection step chooses a test case containing a car as the base test t for mutation, and needs no additional resources. To affect the color change, we manipulate the color in the HSL color space, which separates the components into hue, saturation, and lightness in a way that is similar to human perception of color [14]. At the most basic level, this mutation performs a hue shift on the entity, changing it to a different color. However, this leads to two concerns. First, if we only want to change the color of the vehicle's paint, how do we prevent changing the color of the vehicle's windows? In this color model, high lightness corresponds to white and low to black. This means that since most windows appear dark in images, they will be unaffected by a hue shift. This leads to the second issue: altering the color of white vehicles. To facilitate this, *semImFuzz* checks for high lightness pixels and decreases the lightness and increases the saturation so that the hue shift applied to the entire vehicle produces another color. Specifically, if the average lightness of the vehicle is over 100, then all pixels with a lightness value over 100 have their lightness decreased by a random amount between 20 and 50. Then, to prevent the colors from appearing faded, pixels with a lightness over 100 and a saturation less than 50 have their saturation increased. Figure 6 showcases an example application of this mutation changing the car's color from red to blue. As shown in Figure 6, the change object color mutation can render results that appear very realistic.

4.2.2 Potential for False Positives. As with all mutations, this operation must ensure that r' conforms to some world w'. In general, since r conformed to some world w, then there exists a w' with the vehicle painted the new color. However, the color shift can inadvertently affect other parts of the vehicle and may lead to false positive test cases in which there is no such w'. The very observant reader may have identified in Figure 6 that not only does the car body appear repainted blue but so do the brake lights. In the physical world, it is possible for a car to have blue brake lights; however, this is likely not plausible due to regulations governing the color of the brake light. More advanced versions of this mutation could consider refinements in this area. Another issue, shown in Figure 7, occurs when there is high glare on the vehicle, causing pixelated distortions and thus not conforming with any world w'.

4.3 Adding an Entity

Adding entities to scenes is one of the advances of *semImFuzz*. The goal of this mutation is to add an entity to a scene in a way that may impact the perception system such as adding a vehicle or pedestrian. Adding an entity to the sensor input produces a corresponding addition in the interpretation, and allows us to examine the SUT's performance at the semantic level.

4.3.1 Implementation. For this mutation, the test selection step chooses any test containing a road as the base test t for mutation. The resource query step selects all potential entities of the specified class, e.g. all cars, and filters out those that are occluded by other entities. To increase the likelihood that adding the entity affects the SUT, vehicles must have a bounding rectangle that is at least 100 pixels on its longest side, or 50 pixels for pedestrians. *semImFuzz* then randomly selects an entity and checks if adding that entity to t would result in a conforming test by checking for compatible perspective, semantics, and lighting between the entity and t. These are nontrivial conformity checks, as we explore below.

The resource set provides a collection of images with the pixel boundaries giving the location and outline of isolated entities. When

(a) Source Image (b) Base Image (r)

(c) Isolated Entity in Resources (d) Mutated Image with Car Added (r')

Figure 8: Applying add car mutation (best viewed on a screen)

adding the entity, the mutation takes the isolated portion of the image and overlays it *at the same pixel coordinates* on the base image. Figure 8 demonstrates this process.

Maintaining a consistent perspective is one key to conformity. If done improperly, the added entity will appear out of proportion and misaligned with other image features. To this end, we add the constraint that the entity must be placed at the same relative physical coordinates to the camera in the new test case as it was in the source test case. If we want to add a car with pose p relative to the camera from t_s to test case t to generate t', the resulting t' will consist of t with the car from t_s added at pose p relative to the its camera. Retaining the same pose increases the likelihood that the added car will appear in the correct perspective in t'.

However, adding the entity at the same pixel coordinates does not guarantee that the entity is added at the same physical coordinates relative to the camera. The relationship between pixel coordinates and physical coordinates depends on the characteristics of the camera, the vanishing point of the scene, and the size of the object. We assume that the entire resource set consists of images taken with cameras that have consistent characteristics. Additionally, since the entity is always added so that it occupies the same region of pixels, it has the same size. Thus, if the source and destination images have the same vanishing point, they will be compatible because adding at the same pixel coordinates will result in the same physical coordinates relative to the camera. However, finding pixel-perfect matches on the vanishing point between images is extremely unlikely due to potential number of scenes. Instead, we divide the images into quadrants and consider two images to be compatible if their vanishing points are in the same quadrant. Increasing the precision when matching vanishing points increases the likelihood that the perspective will be adequately maintained, but at the cost of allowing fewer possible mutations and more expense in searching for compatible images during mutation. We utilize an implementation [41] of an algorithm that uses the intersection of Hough lines to determine which region of the image likely contains the vanishing point [22]. If the image that the entity is sourced from and t and do not have compatible vanishing points, it fails the conformity check.

The new location of the entity must be physically feasible in order to conform to a world w'. To validate this, *semImFuzz* checks if the lower left corner of the entity's bounding box is on the road.

Lighting also plays an important role in the conformity of a mutated image, especially the brightness. For example, a car from an image with bright sunlight cannot be added in the shade. To address this, *semImFuzz* takes the region of pixels in the base image that the entity would be placed over and the pixels of the entity and converts them to the HSV (hue, saturation, value) color space. In HSV, the value corresponds to the brightness of the pixels. To ensure that the lighting conditions are similar, if the entity does not have a median value within 5 units ($\sim$ 2%) of the median value of the base image target area, then it fails the conformity check.

Once *semImFuzz* chooses a base image and entity to add, the images are combined into a new test image. Similarly, the interpretation of the base image is edited to include the proper classification of the entity at its position. This is shown in Figures 3e and 3j, where the car has been added both in the image and the interpretation.

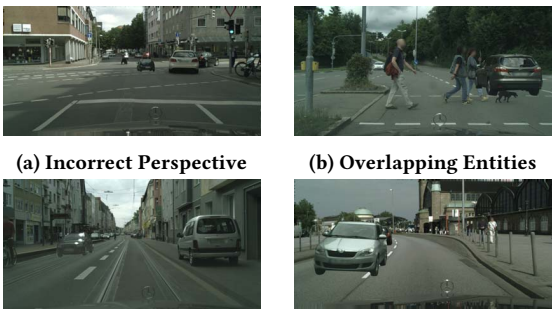
4.3.2 Potential for False Positives. The previous techniques improve the likelihood to generate images with conforming perspective and brightness, but as seen in Figure 9 there are several conditions that can lead to false positives.

While *semImFuzz* takes steps to increase the likelihood of matching consistent perspectives, this does not always happen. Figure 9a shows an example of a car that has been added to an image with a perspective mismatch, causing the added car to appear much smaller than it should compared to the entities around it. Another issue related to entity placement is overlap. Figure 9b shows an instance where a car has been added on top of a person walking their dog. At first it seems that this could have been avoided by checking for overlapping pixels. However, this would overly constrain the mutation and prevent many interesting test cases because pixel-level occlusion does not imply that the entities overlap in the physical world. Determining overlap involves reasoning about the physical sizes of entities and their distance from the camera, which future work could explore.

Another consideration is consistent lighting. In most autonomous vehicle driving scenes, the position of the Sun along with any occlusions like cloud cover determine entity brightness and reflections. For the image to conform to a configuration of the real world, these effects must be consistent with a single lighting configuration. For example, adding a car with a bright reflection to a cloudy image will result in a nonconforming image as shown in Figure 9c. Further, in most lighting conditions all entities in a scene will cast a shadow. Determining the characteristics of the shadows requires a detailed model of the scene with information about the physical location of all entities and light sources. This level of data is not present in the resource set derived from Cityscapes, and so *semImFuzz* does not attempt to add a shadow when adding an entity. This can lead to nonconforming images, as shown in Figure 9d.

4.4 Challenges and Vision for Other Mutations

Currently, *semImFuzz* provides only the aforementioned mutations related to entity recoloring and addition and thus lacks mutations in two key areas: entity repositioning and removal. We now examine where the state-of-the-art in computer graphics and vision is limited to support such advanced mutations, and we comment on our preliminary prototypes and results.



(a) Incorrect Perspective (b) Overlapping Entities

(c) Inconsistent lighting (d) Missing shadow

Figure 9: Nonconforming add car mutations (best viewed on a screen)

Rendering a realistic image of a scene from a specific perspective, is a widely studied area of computer graphics, encompassing all manner of image synthesis. However, the goal of our mutations is to render a realistic image of a scene using real-world data. To do so we need a technique to convert a scene into a rendered image that: (1) is sufficiently novel from prior data, (2) has a known interpretation and (3) conforms to a world w. We find that although state-of-the-art techniques have made strides in all of these areas, there is no technique that meets all three criteria.

4.4.1 Repositioning Entities. Recent machine learning research attempts to render novel scene compositions distinct from that of sampled data. Originally used to render the same entity from a different perspective [23], the current state-of-the-art can remove or slightly reposition entities in the scene [28]. While this approach can render very realistic and conforming images with known interpretation using prior data, it is limited in how "far away" from the original scene it can operate. Entities can only be removed if they did not occupy that space in a different video frame and repositioning is limited to a few meters and degrees different than the original scene. While this area is promising, it currently does allow for sufficient variation from prior data to use in *semImFuzz*.

4.4.2 Removing Entities. Another relevant thread of graphics research is the problem of inpainting, the task of filling in regions of an image such that the new image is conforming [20, 39]. For example, removing a car from the scene can be achieved by inpainting over the region of the car with empty road. Although inpainting has potential to produce conforming images for use in mutation, in practice such tools satisfy neither our interpretation nor conformity requirements. The interpretation of the inpainted region is not known and although the system is trained on real data, its ability to produce conforming images decreases as the inpainted region grows. We developed a prototype using [39] to perform a mutation to remove a car from an image. Figure 10 shows an example output of using inpainting to remove a car from an image taken from the nuScenes [5] data set. As shown in Figure 10b, although the car itself is no longer visible, the region exhibits several nonconforming distortions and lighting effects which hint at the prior existence of the car. Advancements in inpainting to make the procedure more robust would greatly expand the space of feasible mutations.

(a) Original Car (b) Car Inpainted

Figure 10: Inpainting to Remove Entities (best viewed on a screen)

4.4.3 Direct Image Synthesis. Direct image synthesis systems work as the inverse to the perception system of an AS, taking in an annotated interpretation of an image and rendering a camera image that is consistent with that interpretation [29]. This would remove the need to perform mutations on the sensor inputs, allowing *semImFuzz* to mutate the interpretation directly and use direct image synthesis to create a matching sensor reading. This process can richly combine data to produce novel scenes, but current implementations still do not satisfy the interpretation and conformity requirements. While interpretation is considered during synthesis, some regions are left unconstrained and are delegated to inpainting, suffering from the shortcomings outlined above.

4.4.4 Discriminators for Conformity Checking. In prior sections, we describe the possibilities for false positives arising from nonconforming test cases. Machine learning discriminators present a way to identify such nonconforming tests [25]. These discriminators are a form of binary classifier that seek to determine if an input belongs to a distribution, so they could preemptively reject mutations that are nonconforming with the real world defined by a training set. We explored this approach, training a binary classifier based on a CNN using over 48,000 images taken from both the Cityscapes data set and those generated by *semImFuzz*. The discriminator learned to differentiate between the classes, but could not differentiate conforming versus nonconforming images. In part, the problem is that the generated images contain conforming and nonconforming images, but checking 24,000 of them was prohibitively expensive. More generally, a broader challenge is that discriminators tend to require much larger data sets even for much smaller images. We suspect that the discriminator requires much more training data to target a resolution suitable for ASs, which is not feasible due to the cost to obtain real-world data. Future research in this direction has the potential to reduce the false positive rate of *semImFuzz*.

5 DISCUSSION OF TRADE-OFFS

semSensFuzz provides a framework for testing AS perception systems distinct from the prior approaches described in Section 2. We note here that these prior approaches are not suitable for direct comparison due to the differing goals and intended outputs. Simulation, low-level mutation, and *semSensFuzz* provide different trade-offs in their ability to generate novel outputs, cost to generate tests, and conformity of outputs. We now discuss those trade-offs.

In terms of ability to generate novel inputs, simulation provides the highest level of utility, allowing for practically unlimited environment generation. At the other end, low-level mutation techniques provide a narrow range of possible test cases based on prior data and the global mutation strategies employed, being limited

in the novelty of the mutated data compared to the original data. *semSensFuzz* builds from the low-level mutation strategy in terms of utility, providing new dimensions for the novelty of the mutated data. Although still limited by the availability of prior data, the richer semantic mutation strategies allow for the generation of more meaningful novel test cases that contain a substantive and human-understandable semantic change.

Although simulation provides the highest level of utility for generating novel inputs, its efficiency is more nuanced. Simulated environments may be re-used between testing different systems, but building a new environment comes at a high cost in terms of time and expertise required. Once an environment is in place for testing, adapting the environment can be achieved at a lower cost. For example, given an existing scenario, the simulator CARLA [8] provides an automated way to vary weather conditions. For both low-level mutation and *semSensFuzz*, there is an initial cost to collect the baseline data to mutate; however, readily available data sets [5, 7, 11] can ameliorate these costs to the end-user. Using available data, the low-level mutation strategies are likely the most efficient in producing new test cases. Given the additional machinery that *semSensFuzz* uses to leverage multiple facets of the prior data while also performing conformity checks, the computational cost is likely significantly higher than the low-level approaches.

As discussed in Section 2, one of the key limitations of simulation testing is the simulation-reality fidelity gap that can diminish the applicability of the tests and require revalidation in the physical world as they may not be conforming with any real world [15, 21, 40]. The mutation-based strategies are designed to avoid this limitation by using real-world data as the baseline and ensuring that the mutations preserve the conformity of the original data. We note that *semSensFuzz* specifically identifies this need to preserve conformity as a fundamental part of the framework.

ASs are complex systems that require a varied complement of testing techniques. The validation process for any AS should include multiple types of testing. *semSensFuzz* provides a valuable addition to the validation toolbox by providing semantic mutations absent in the low-level mutation strategies, while maintaining conformity to ensure it avoids the pitfall of the simulation-reality gap.

6 EVALUATION

We have developed *semImFuzz*, a test generation tool that can efficiently identify performance inconsistencies in AS perception systems using semantic mutations. To evaluate the potential of our approach, *semImFuzz*, in these dimensions we seek to answer the following research questions:

RQ1. How effective is our technique in uncovering inconsistencies defined at different levels of severity?

RQ2. How efficient is our technique in terms of the time taken to generate the mutation and to detect the inconsistencies?

RQ3. Which mutations are the most effective?

6.1 SUTs

To assess *semImFuzz*, we evaluate it on five highly competitive perception systems submitted to the Cityscapes benchmark for the "Pixel-Level Semantic Labeling Task" [7]. These SUTs were the five highest performing SUTs for which code and pre-trained models were publicly available. Each project was forked from the original repository, edited if necessary to provide a consistent output format, and packaged to run in a Docker container for replicability. Table 1 lists SUTs ranked by their performance (overall and among the ones with code) on the original Cityscapes dataset as per the default IoU Class metric. Intuitively, this is the percentage of correctly labeled pixels, with a minimum score of 0 and a maximum of 100. Pixels that do not belong to a class are marked as "do not care" (like the hood of the ego vehicle) meaning that they are excluded from scoring regardless of the SUT assigned label.

Table 1: Cityscapes benchmark scores for SUTs evaluated

Rank		Year	SUT	IoU
Overall	With Code			
5	1	2020	NVIDIA SemSeg [34][2]	85.4
17	3	2021	EfficientPS [24][3]	84.2
23	6	2020	DecoupleSegNet [19][4]	83.7
26	7	2018	SDCNet [44][5]	83.5
29	9	2019	HRNetV2+OCR [33][6]	83.3

6.2 Tests Generated

We used the Cityscapes data set as the basis for our testing, serving as the original test suite and basis for building the resource set. We pruned test cases that are too difficult for the SUTs to prevent mutating test cases where the mutation will not be the focus of the test; if the SUT struggles with the image as a whole, the mutation will not provide any additional utility. To perform this filter, we ran the highest performing SUT on the baseline, as determined by its ranking on the Cityscapes leaderboard [7], on the original test cases in the data set to establish the baseline performance of each test case. Any test case on which the best SUT performs below the threshold is removed from consideration. For our testing, we set this minimum score parameter at 95% resulting in the removal 42 of the total 3,475 images in the data set. *semImFuzz* then performs a one-time preprocessing analysis of the remaining test cases, as described in Section 4, to build the resource set by finding all available entities across the test cases to include in future mutations.

For evaluation, we ran each SUT on 150,000 tests generated by *semImFuzz* comprised of 50,000 "Add Car" mutations, 50,000 "Add Person" mutations, and 50,000 "Change Car Color" mutations.

6.3 Metrics

An effective mutation, one that finds a potential SUT failure mode, renders a conforming image that causes the SUT to perform poorly. To capture that notion we rely on the Cityscapes benchmark evaluation tool (*eval*) which scores each SUT's performance based on the percentage of correctly classified pixels. We evaluate the performance of the mutation strategies by calculating the percentage point (p.p.) difference between the SUT's score on the original image and its score on the mutated image. Any drop in the SUT's

[2]https://github.com/NVIDIA/semantic-segmentation/tree/7726b14
[3]https://github.com/less-lab-uva/EfficientPS
[4]https://github.com/less-lab-uva/DecoupleSegNets
[5]https://github.com/less-lab-uva/semantic-segmentation/tree/sdcnet
[6]https://github.com/less-lab-uva/HRNet-Semantic-Segmentation

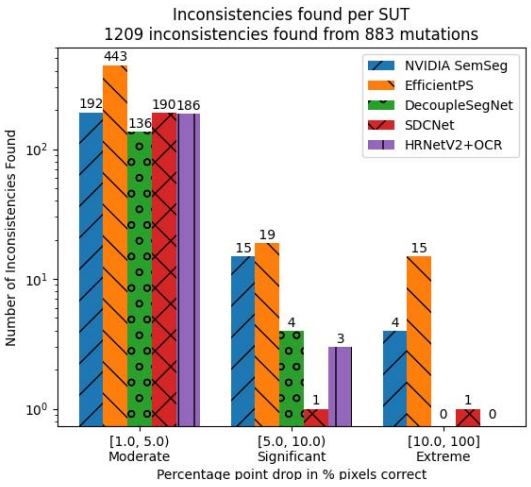


Figure 11: Inconsistencies found per SUT

score on the mutated image compared to the original image was induced directly by the mutation itself, allowing us to gauge the strength of the mutations to find inconsistencies.

More formally, for a given SUT *Percep*, test case $(r, interp)$, and corresponding mutation $(r', interp')$, the drop is given by

$$drop = eval(Percep(r), interp) - eval(Percep(r'), interp')$$

The larger the value, the more error the mutation induced in the SUT. We judge drops of less than 1 p.p. to be in the noise as most tests result in SUTs misclassifying a few pixels around the edges of objects, leading to small drops compared to the baseline. We select 1 p.p. as the cut off to reduce the likelihood that we deem an inconsistency noteworthy when it is not, and as a filter to maintain the feasibility to complete the manual reviews we conduct for evaluation. We categorize drops between 1 and 5 p.p. as moderate inconsistencies, between 5 and 10 as significant, and those greater than 10 as extreme inconsistencies. For additional context, the hood of the car visible at the bottom of all images in the Cityscapes data set (see Fig. 3), occupies roughly 4.3% of the image; a drop of more than 5 p.p. means an area larger than the hood was misclassified. We note that there is no additional filtering at this step based on how well the SUT performed on the original test case. For example, if the SUT scored only 85% on the original test and then scored 78% on the mutated test, then this would result in a drop of 7 p.p. and be classified as a significant inconsistency. Future work may examine using specific performance thresholds instead of performance drops, specifically as this technique applies to testing a single SUT. However, by measuring deterioration of performance, we are able to use this metric to compare across the various SUTs even though they show different levels of absolute performance.

We also measure the time to perform each mutation and the time to run the SUTs to assess efficiency.

6.4 Results

6.4.1 RQ 1 Results: Finding Inconsistencies. Figure 11 shows the counts of inconsistencies found in each category per SUT; note the log scale on the y-axis. We first remark that *semImFuzz* found

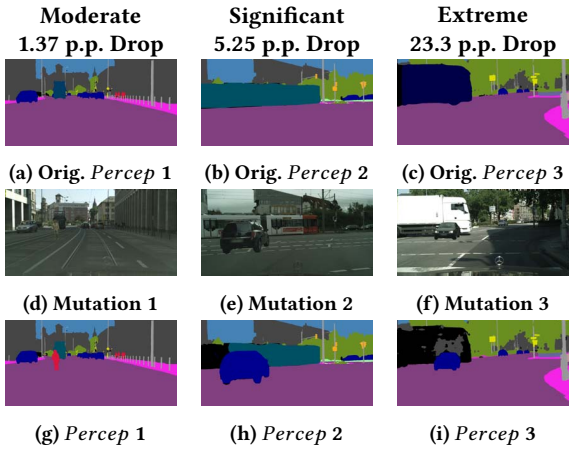


Figure 12: Visualizing SUT Inconsistencies across Severities (best viewed on a screen)

1210 SUT inconsistencies resulting from 884 mutations and that each SUT had over 100 inconsistencies. Further, each of the SUTs exhibited at least one significant inconsistency and 3 of the 5 SUTs combined to exhibit a total of 20 extreme inconsistencies.

The distribution of inconsistencies among the SUTs is unexpected. NVIDIA SemSeg [34] and EfficientPS [24], the two highest scoring on the Cityscapes benchmark, had the highest number of inconsistencies in all three categories with 211 and 477 respectively. EfficientPS revealed more than three times the number of inconsistencies of the SUT with the fewest inconsistencies.

For contextualizing the magnitude of the inconsistencies, Figure 12 shows three mutations produced by *semImFuzz* and their interpretations by the EfficientPS SUT before and after the mutation. The leftmost column shows a moderate inconsistency; an added person occludes a bus, causing EfficientPS to then classify the bus as a train. The middle column shows a significant inconsistency; an added car occludes a train, causing EfficientPS to correctly identify only part of the train, labeling the rest as "do not care". The rightmost column shows an extreme inconsistency; an added car occludes a truck, causing EfficientPS to misclassify the truck, labelling portions as "do not care" and the rest as building.

We also assess effectiveness of *semImFuzz* in terms of inconsistencies found over time. Figure 13 shows the number of significant and extreme inconsistencies found versus the number of mutated tests executed. While *semImFuzz* generated 150,000 tests, the graph shows the average of 10 permutations to control for parameter selection randomness. Figure 13 shows that the number of inconsistencies found continues to increase even at 150,000 tests, indicating that fuzzing has not saturated. Again we note that the two strongest SUTs on the benchmark yield significant and extreme inconsistencies at a much higher rate than the other SUTs. Further analysis of the specific SUTs is needed to understand the factors involved in this performance, but these data suggest that the highest performing SUTs may be more brittle under certain conditions.

Since *semImFuzz* uses random test and resource selection, we computed the number of duplicate tests generated as an indication of saturation. Only 1228 (0.82%) of the 150,000 tests were duplicates,

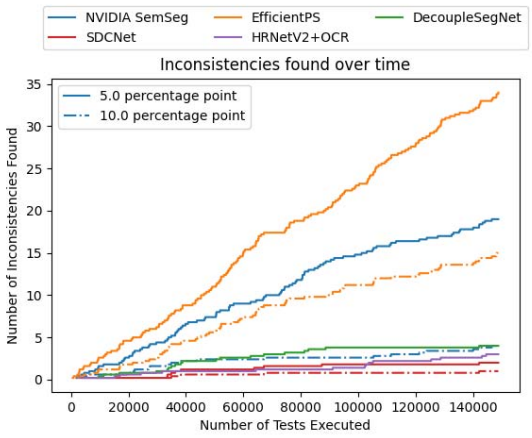


Figure 13: Inconsistencies found over time

Table 2: False Positive Rate for Inconsistencies Found

SUT	False Positive Rate		
	[1, 5)	[5, 10)	[10, 100]
NVIDIA SemSeg [34]	11%	67%	50%
EfficientPS [24]	43%	47%	53%
DecoupleSegNet [19]	38%	75%	—
SDCNet [44]	42%	0%	0%
HRNetV2+OCR [33]	28%	67%	—

suggesting that fuzzing was not near saturation. This further highlights the ability of our approach to generate orders of magnitude more data than the original test suite which contained 3,433 tests suitable for mutation rendering a more than 40-fold increase.

We have shown that *semImFuzz* can generate test cases that induce inconsistencies in the SUTs. However, one potential issue is the presence of test cases that are nonconforming, which leads to false inconsistencies. As highlighted in Section 4, there are several factors that can result in a nonconforming test case, and determining conformity is subjective. Still, to gain a better grasp on the rate of false positives we manually inspect all the generated test cases that led to 5+ p.p. drop, and sample 10% of the test cases that led to a drop between 1 and 5 p.p.. The process entailed each author examining each image and classifying them as either a true positive or false positive. If any of the three authors deemed an image a false positive, it was conservatively recorded as such.

To further convey the quality of our assessment, Figure 14 showcases 5 true positive and 5 false positive test cases that induced inconsistencies in our study. The false positive rates are shown in Table 2. The high false positive rate reflects the challenging application domain to achieve conformity. Still, this high rate is mitigated by the number of instances on which it applies. Generating 150,000 test cases produces a few hundred tests with inconsistencies of interest which developers can examine to understand performance or select future testing directions. In this examination, determining if an image is nonconforming takes a few seconds of time, meaning encountering a false positive incurs a relatively low cost. Furthermore, the number of inconsistencies found per SUT is small and can be prioritized by the drop measure.

6.4.2 RQ 2 Results: Efficiency. We consider efficiency by comparing the time for *semImFuzz* to mutate a test with the time for the SUTs to run a test. Table 3 shows the average time to generate each type of mutation (2nd row) compared to the average time to run each SUT on those tests (3rd-7th row) in milliseconds, with standard deviation in parentheses. For each table cell, we generated 100 tests 10 times, and averaged the times across these trials.

The color change mutation was the fastest, taking less than 20% the time of the add mutations. This is because the conformity check for the color change mutation always passes, so it spends less time selecting viable resources. The add car mutation was slightly faster than the add person mutation. This is likely because it is easier to satisfy the conformity constraints—it is more likely for a randomly sampled car to appear on the road than a randomly sampled person, meaning fewer samples are needed to find a suitable car. As expected, we found that for each SUT the time it takes to execute the test is not different based on the mutation. While the fastest SUT, HRNetV2+OCR [33], takes about as much time to execute a test as *semImFuzz* does to generate a test, we note that three of the SUTs take more than double that time, and one, NVIDIA SemSeg [34], takes more than 5 times as long. Further, once a set of mutations have been generated, they can be used in the future to test additional revisions of the SUT at no additional cost to prepare.

Table 3: Average time to generate and execute a test in milliseconds, with the standard deviation in parenthesis.

Activity	Add Car	Add Person	Color Car
Test Generation	593 (19.9)	642 (42.7)	105 (3.06)
NVIDIA SemSeg [34]	3421 (29.5)	3418 (22.9)	3404 (31.5)
EfficientPS [24]	837 (7.22)	837 (5.88)	837 (7.41)
DecoupleSegNet [19]	1426 (2.24)	1425 (2.60)	1423 (2.48)
SDCNet [44]	1328 (3.09)	1328 (2.81)	1327 (8.33)
HRNetV2+OCR [33]	600 (5.89)	605 (4.23)	605 (2.89)

6.4.3 RQ 3 Results: Mutation Types. In this section we examine more carefully the results per mutation type. Figure 15 shows the inconsistencies found for each SUT based on the mutation type; note the log scale on the y-axis. The add car mutation induced the most inconsistencies (877), followed by the add person mutation (280), and then the mutation to change the car color (53). These results reflect what we may expect; adding a car affects a large portion of the image, leading to a higher likelihood of the mutation yielding an inconsistency. However, even though recoloring a car affects the same number of pixels as adding that car to another image, we find that the SUTs are robust against changing the color of the car. This suggests that editing large regions of the image, even in a conforming manner, is insufficient. This further supports our notion that high level semantic mutations such as adding a car are required to exercise these perception systems and find inconsistencies.

6.5 Threats to Validity

The external validity of our findings for *semImFuzz* are affected by our choice of data set and SUTs. We selected Cityscapes for its

(a) True Positive 1 (b) True Positive 2 (c) True Positive 3 (d) True Positive 4 (e) True Positive 5

(f) False Positive 1 (Overlapping) (g) False Positive 2 (Lighting) (h) False Positive 3 (Incorrect Coloring) (i) False Positive 4 (Perspective & Overlap) (j) False Positive 5 (Incomplete Entity)

Figure 14: Sample of True and False Positives (best viewed on a screen)

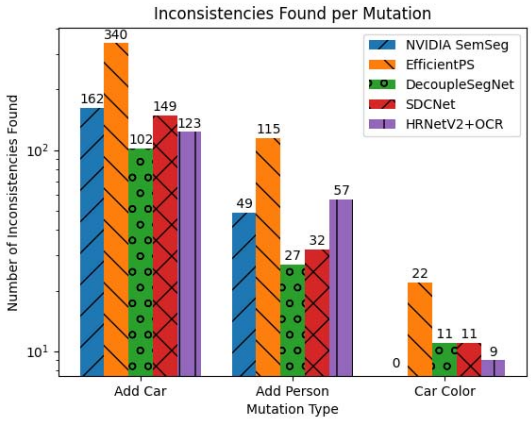


Figure 15: Inconsistencies by mutation type

popularity as a benchmark of perception systems, and for SUTs we selected the top five that made their source available for reuse in the Cityscapes competition. Extending the scope to data sets such as nuScenes [5], KITTI [11], and Waymo [9] would help generalize the findings. More broadly, the proposed approach is more general than *semImFuzz*, being applicable to other sensor data such as the commonplace LIDAR (light detection and ranging) for remote sensing. This level of generalization remains to be tested empirically.

The internal validity of our findings may be affected by several factors. Top among them is the implementation of the mutations, which is complex, and includes external components and many parameters. In spite of our validation efforts, they could have faults. We share the code to mitigate that threat. Also, by restricting the initial tests to those on which the SUTs did well we helped isolate the effect of the mutations. However, this constraint may have left out opportunities to make tests that render poor results even worse. We also attempted to control for the randomness of several nondeterministic components through repeated executions.

In terms of construct validity, although we have quantitatively shown that *semImFuzz* can generate many test cases that yield inconsistencies, our examination of false positives for conformity exhibits inherent bias. We share the code to reproduce the test suite to mitigate this threat. Further, the inconsistencies we found in terms of the percentage point drop may not extend to cause failures in real ASs. While Section 6.4 showcases several serious inconsistencies, further study is needed to understand if and how these inconsistencies would affect the entire AS.

7 CONCLUSION AND FUTURE WORK

We introduced a novel approach that automatically generates test-cases with sensor reading and ground-truth interpretation pairs for AS perception systems. The approach leverages domain-specific semantics and prior test cases based on real-world sensor data to generate mutated sensor readings that still conform to the physical world. Our experimental prototype for images showed that low-cost and high-level semantic mutations such as adding a car can uncover inconsistencies in state-of-the-art perception systems.

semSensFuzz and our implementation *semImFuzz* set several directions for future work. Following standard software fuzzing, we will examine how to use SUT performance to guide mutation type and parameter selection to more quickly find inconsistencies. We will also investigate how to expand to other sensor modalities and aggregate readings from multiple sensors to more holistically test AS perception systems. This work also encourages several directions of research in computer graphics, advances in which would likely translate quickly into improvements in *semImFuzz*.

Testing AS perception systems *requires* examining rare, safety-critical scenarios which cannot be obtained practically from the real world. We will develop more advanced mutations that incorporate these elements, such as cars driving in the wrong direction, damaged cars, or people sitting on cars. Although the current implementation has the *opportunity* to generate some of these elements, e.g. a car driving the wrong direction, making such mutations an explicit design goal will require the integration of richer data sets and the development of more sophisticated mechanisms to check conformity. The ability to systematically generate these scenarios will further bolster the utility of our approach by generating conforming sensor readings of events that are impractical to otherwise obtain and have exhibited problems for real-world deployed ASs.

ACKNOWLEDGEMENTS

This work was supported in part by funds provided by NSF#1924777 and NSF#1909414. Trey Woodlief was supported by a University of Virginia SEAS Fellowship. We are thankful to David Luebke of NVIDIA for his help in understanding the state-of-the-art in computer graphics and vision.

REFERENCES

[1] 2019. Uber in fatal crash had safety flaws say US investigators. *BBC* (Nov 2019). https://www.bbc.com/news/business-50312340

[2] Adith Boloor, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. 2019. Simple physical adversarial examples against end-to-end autonomous driving models. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 1–7.

[3] Neal E Boudette and Niraj Chokshi. 2021. U.S. Will Investigate Tesla's Autopilot System Over Crashes With Emergency Vehicles. *New York Times* (Aug 2021). https://www.nytimes.com/2021/08/16/business/tesla-autopilot-nhtsa.html

[4] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[5] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. 2019. nuScenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027* (2019).

[6] Alex Clark. 2015. Pillow (PIL Fork) Documentation. https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*. PMLR, 1–16.

[9] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles Qi, Yin Zhou, Zoey Yang, Aurélien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. 2021. Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset. *arXiv preprint arXiv:2104.10133* (2021).

[10] Xiang Gao, Ripon K Saha, Mukul R Prasad, and Abhik Roychoudhury. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1147–1158.

[11] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)* (2013).

[12] Isobel Asher Hamilton. 2019. Tesla is being sued again for a deadly Autopilot crash. *Insider* (Aug 2019). https://www.businessinsider.com/tesla-sued-family-jeremy-beren-banner-autopilot-crash-2019-8

[13] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[14] Noor A Ibraheem, Mokhtar M Hasan, Rafiqul Z Khan, and Pramod K Mishra. 2012. Understanding color models: a review. *ARPN Journal of science and technology* 2, 3 (2012), 265–275.

[15] Nick Jakobi, Phil Husbands, and Inman Harvey. 1995. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*. Springer, 704–720.

[16] Nidhi Kalra and Susan M. Paddock. 2016. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, Santa Monica, CA. https://doi.org/10.7249/RR1478

[17] Zelun Kong, Junfeng Guo, Ang Li, and Cong Liu. 2020. Physgan: Generating physical-world-resilient adversarial examples for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14254–14263.

[18] Tom Krisher. 2018. Feds: Tesla accelerated, didn't brake ahead of fatal crash. *AP News* (Jun 2018). https://apnews.com/article/north-america-us-news-mi-state-wire-ca-state-wire-transportation-8c833b3e5d9c49cf97a10974126daad9

[19] Xiangtai Li, Xia Li, Li Zhang, Cheng Guangliang, Jianping Shi, Zhouchen Lin, Yunhai Tong, and Shaohua Tan. 2020. Improving Semantic Segmentation via Decoupled Body and Edge Supervision. In *ECCV*.

[20] Liang Liao, Jing Xiao, Zheng Wang, Chia-Wen Lin, and Shin'ichi Satoh. 2020. Guidance and evaluation: Semantic-aware image inpainting for mixed scenes. In *European Conference on Computer Vision*. Springer, 683–700.

[21] Waymo LLC. 2020. *Waymo's Safety Methodologies and Safety Readiness Determinations*. Technical Report. 30 pages. https://storage.googleapis.com/sdc-prod/v1/safety-report/Waymo-Safety-Methodologies-and-Readiness-Determinations.pdf

[22] Andrea Matessi and Luca Lombardi. 1999. Vanishing point detection in the hough transform space. In *European Conference on Parallel Processing*. Springer, 987–994.

[23] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*. Springer, 405–421.

[24] Rohit Mohan and Abhinav Valada. 2020. EfficientPS: Efficient Panoptic Segmentation. *International Journal of Computer Vision* 129 (2020), 1551 – 1579.

[25] Huy H Nguyen, T Ngoc-Dung Tieu, Hoang-Quoc Nguyen-Son, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. 2018. Modular convolutional neural network for discriminating between computer-generated images and photographic images. In *Proceedings of the 13th international conference on availability, reliability and security*. 1–10.

[26] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. PMLR, 4901–4911.

[27] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, Russ Tedrake, and John C Duchi. 2018. Scalable end-to-end autonomous vehicle testing via rare-event simulation. *Advances in neural information processing systems* 31 (2018).

[28] Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. 2021. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2856–2865.

[29] Xiaojuan Qi, Qifeng Chen, Jiaya Jia, and Vladlen Koltun. 2018. Semi-parametric image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8808–8816.

[30] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. 2018. Semantic foggy scene understanding with synthetic data. *International Journal of Computer Vision* 126, 9 (2018), 973–992.

[31] Hans-Peter Schöner. 2018. Simulation in development and testing of autonomous vehicles. In *18. Internationales Stuttgarter Symposium*. Springer, 1083–1095.

[32] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering* 42, 9 (2016), 805–824.

[33] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. 2019. Deep High-Resolution Representation Learning for Human Pose Estimation. In *CVPR*.

[34] Andrew Tao, Karan Sapra, and Bryan Catanzaro. 2020. Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821* (2020).

[35] Brad Templeton. 2019. NTSB Report On Tesla Autopilot Accident Shows What's Inside And It's Not Pretty For FSD. *Forbes* (Sep 2019). https://www.forbes.com/sites/bradtempleton/2019/09/06/ntsb-report-on-tesla-autopilot-accident-shows-whats-inside-and-its-not-pretty-for-fsd/?sh=6270d8234dc5

[36] Brad Templeton. 2020. Tesla In Taiwan Crashes Directly Into Overturned Truck, Ignores Pedestrian, With Autopilot On. *Forbes* (Jun 2020). https://www.forbes.com/sites/bradtempleton/2020/06/02/tesla-in-taiwan-crashes-directly-into-overturned-truck-ignores-pedestrian-with-autopilot-on/?sh=20a7458f58e5

[37] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[38] Christopher Steven Timperley, Afsoon Afzal, Deborah S Katz, Jam Marcos Hernandez, and Claire Le Goues. 2018. Crashing simulated planes is cheap: Can simulation detect robotics bugs early?. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 331–342.

[39] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. 2018. Generative image inpainting with contextual attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5505–5514.

[40] Juan Cristóbal Zagal and Javier Ruiz-Del-Solar. 2007. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems* 50, 1 (2007), 19–39.

[41] Sebastian Zanlongo, Matthew Turk, and Sanjay Parajuli. 2019. vanishing-point-detection. https://github.com/SZanlongo/vanishing-point-detection.

[42] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. 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)*. IEEE, 132–142.

[43] Husheng Zhou, Wei Li, Zelun Kong, Junfeng Guo, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 347–358.

[44] Yi Zhu, Karan Sapra, Fitsum A Reda, Kevin J Shih, Shawn Newsam, Andrew Tao, and Bryan Catanzaro. 2019. Improving semantic segmentation via video propagation and label relaxation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8856–8865.