Generating Realistic and Diverse Tests for LiDAR-Based Perception Systems

Abstract—Autonomous systems rely on a perception component to interpret their surroundings, and when misinterpretations occur, they can and have led to serious and fatal system-level failures. Yet, existing methods for testing perception software remain limited in their capacity to efficiently generate test data that is both realistic in order to translate to real-world performance and diverse in order to capture the long tail of rare but safety-critical scenarios. These limitations are particularly evident for perception systems based on LiDAR sensors, which have emerged as a crucial component in modern autonomous systems due to their ability to provide a 3D scan of the world and operate in all lighting conditions. To address these limitations, we introduce a novel approach for testing LiDAR-based perception systems by leveraging existing real-world data as a basis to generate realistic and diverse test cases through mutations that preserve realism invariants while generating inputs rarely found in existing data sets, and automatically crafting oracles that identify potentially safety-critical issues in perception performance. We implemented our approach to assess its ability to identify perception failures, generating over 50,000 test inputs for five state-of-the-art LiDAR-based perception systems. We found that it efficiently generated test cases that yield errors in perception that could result in real consequences if these systems were deployed and does so at a low rate of false positives.

Index Terms—Software Testing and Validation, Machine Learning

I. INTRODUCTION

As autonomous system (AS) research and development accelerates, so too must our efforts to rigorously test them to reduce field failures. A key component underpinning all aspects of ASs is the perception software that transforms sensor inputs into an interpretation of the world [16]. Since all actions of the AS are predicated on its world interpretation, perception failures can lead to safety-critical system-level failures. For example, in 2018 an Uber autonomous vehicle going 35 mph at night collided with a woman crossing the street with her bike [15], [17], [30]. The perception component consuming a LiDAR point cloud (PC), a vector containing a set of locations in 3D denoting how far away an object is from the sensor (as exemplified in Figure 1a), produced an interpretation of the bike that oscillated between vehicle, "other", and bike in the 10 seconds leading up to the crash [4]. Only 1.2 seconds before the collision the component produced a consistently correct interpretation, which was too late to alert the human driver or brake, resulting in a fatal collision.

Field failures in AS perception components have resulted in multiple fatalities in recent years [4], [5], [41]. Exposing these failures, however, can be particularly challenging when the input is provided by LiDAR. While LiDAR is becoming more prevalent in autonomous vehicle companies (e.g., Volvo [29],

(a) Raw Point Cloud   (b) Expected Segmentation

(c) Raw Range Image

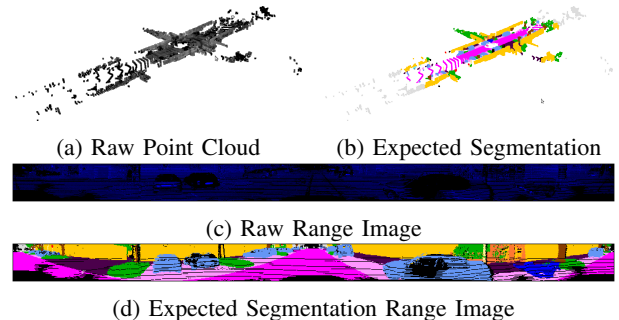(d) Expected Segmentation Range Image

Fig. 1: Sample Point Cloud with Intensities and Expected Segmentation (best viewed on a screen)

Motional [3], Baidu [50], Ford [1], Zoox [34], and Waymo [33]), our approaches to generate realistic and diverse inputs to test these components remain quite limited for several reasons.

First, although real-world LiDAR PCs can be collected while driving, generating their expected outputs through manual labeling can be prohibitively expensive as PCs are difficult to label, requiring significant human effort along with specialized training and software, limiting availability to a few data sets [2], [6], [9], [21], [32], [35], [39]. As an example of the difficulty in labeling, Fig. 1 shows a sample PC presented as a 3D scene in Fig. 1a and as a 2D projection in Fig. 1c with their respective expected semantic labels in Figs. 1b and 1d. While large objects such as vehicles may be recognizable by their shape, discerning labels for other points is challenging due to the disconnect between PCs and how humans perceive the world—we have the analog of built-in cameras, but not built-in LiDAR. Besides labeling challenges, these data sets are inherently limited in diversity as the collected data is unlikely to include dangerous edge cases such as a person crossing directly in front of the vehicle. This rarity is a fundamental limit of data collection efforts as it is unfeasible, uneconomical, and unethical to collect the safety-critical edge cases necessary for sufficient diversity in testing.

Second, although simulation-based techniques can serve to generate synthetic LiDAR inputs and use the simulator state to compute the expected outputs [12], [19], [44], [52], the simulation-reality domain gap [25] makes it difficult to judge whether a system's performance is due to the inherent synthesised nature of the data or due to component failures [27], [44]. In other words, is a failure a false positive caused by unrealistic test data? In spite of recent advances in LiDAR simulation techniques to mitigate this challenge

by supplementing simulators with real-world PCs [28], [46], the simulation-reality gap remains a significant challenge. Although simulation allows for improved test diversity, it is limited by the level of realism, leading to false positives.

Finally, LiDAR-specific data generation techniques have not focused on the specific need to generate realistic data from the long tail of possible scenes in order to increase diversity. The two closest lines of work are as follows. First are techniques for generating adverse weather effects [22], [26] and whole-PC affine transformations [20] that, while realistic, provide limited increases in diversity as they do not change expected output at the semantic level, i.e. the semantics of each point is the same even if the point is moved or obscured. Second are techniques for adding objects to a scene [14], [49] that have the potential to increase diversity by altering scene semantics, but do not ensure realism. Section V-B contains a full discussion of related data augmentation techniques.

Recent work on testing image-based perception systems proposed using existing real-world images and performing mutations at the semantic level to alter both the input and expected output [47]. Similar to our aim, this work considers realism and diversity as first order concerns, generating realistic tests that are often lacking in existing data sets. Although this approach articulated a high-level vision, the mutations described and studied only explored the applicability to image-based perception systems and do not naturally translate to LiDAR-based systems.

Inspired by the work on semantic mutation testing with images, this work performs the non-trivial extension to generate test cases for LiDAR-based perception systems, aiming to generate test cases that can explore the long tail of possible scenes, such as a woman walking with her bike in front of the AS. Different from images, semantic mutations for LiDAR require a reimagining of the input mutation space, with special considerations for the physics of the LiDAR sensor and the 3D nature of the input domain, and specialized techniques for evaluating the performance of the system under test (SUT) as even small failures such as misclassifying a pedestrian that is 0.05% of the total points can be safety-critical.

Our approach pursues realism by starting with training data consisting of real-world PCs that have been semantically labeled, tapping into this real data to build resources that can be used in mutation operations, and by designing domain-specific mutations that respect real-world invariants. Our approach pursues diversity by developing mutations that can complement the available real-world data sets to produce diverse test inputs that could be found in real life situations. Then, by using metamorphic relationships between the original and mutated data, we can find potentially dangerous failures in perception that are not covered by the original data set. Further, motivated by the inherent difficulty for humans to perform the LiDAR perception task, this work proposes an automated mechanism for evaluating the realism of the mutations; by examining the performance of multiple SUTs, we can automatically identify potential false positives from unrealistic inputs. Our contributions are as follows:

- A new approach for testing LiDAR based perception systems, based on semantic mutations applied to labeled real-world data.
- An implementation targeting LiDAR perception of autonomous vehicles, publicly releasing the code[1].
- Experiments using our approach to test five state-of-the-art LiDAR perception systems, revealing errors in perception that could result in real consequences if these systems were deployed.

II. APPROACH

Our approach takes as input a data set of LiDAR point clouds collected from the real world and labeled with ground truth semantics, and performs realistic semantic mutations to generate a diverse set of novel test cases that meaningfully exercise LiDAR perception systems. We respect realism by starting with real-world tests, collecting entities that already reside in the real world as resources, and crafting realism invariants that must hold during the mutation process. Further, we increase diversity by crafting mutations that can render hard to find cases, such as a car driving the wrong way on a street or a pedestrian in the roadway.

Our approach is divided into three stages which we discuss in detail in this section: Resource Collection, Mutation Generation, and Performance Evaluation.

A. Preliminaries

LiDAR, or **L**ight **D**etection **A**nd **R**anging, technology measures the round-trip time-of-flight of a beam of light between the sensor and an object to determine how far away the object is from the sensor, analogous to radar using sound waves [45]. Many LiDAR systems used in ASs leverage a rotating sensor that allows the system to take discrete measurements at regular intervals to obtain the distance to many different objects within a scene. Taken together, these measurements form a *point cloud* (PC), a representation of the sensor reading that uses the sensed distances and calculated angles to create set of points in 3D space that depict the locations of all sensed objects.

In many ASs, the LiDAR serves as an input to the perception software, leveraging the PC to perceive the semantics of the scene. We consider the semantic segmentation task in which a perception system assigns each point in a PC a semantic class identifying what the point represents, e.g. road, person, car, etc. Fig. 1 shows an example of a LiDAR PC used in perception, Figs. 1c and 1d show a range image where the 360° view is projected onto a cylinder and unrolled. Figs. 1a and 1c are colored with the corresponding reflected intensities of the LiDAR beams. Figs. 1b and 1d are colored with the human-annotated semantic class of each point.

A test case for a LiDAR-based perception system consists of an input PC, PC, an expected output semantic labeling, $ExpSem$, and an evaluation function, $eval$. The labeling from the SUT, $SUT(PC)$, is evaluated against the expected output using the evaluation function, i.e. $eval(ExpSem, SUT(PC))$ to determine if the SUT passes the test.
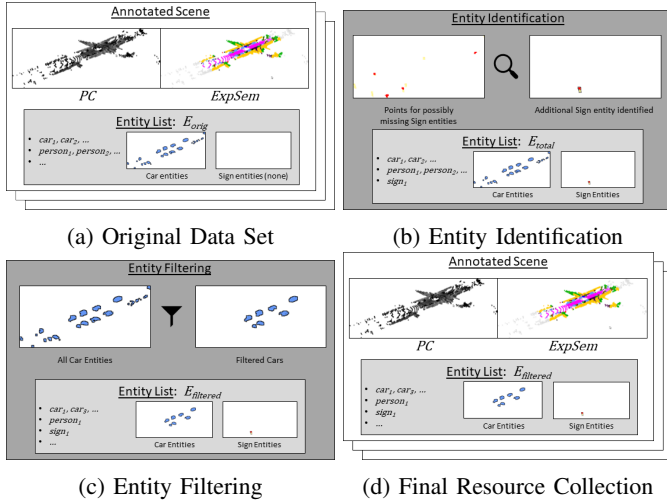
(a) Original Data Set   (b) Entity Identification

(c) Entity Filtering   (d) Final Resource Collection

Fig. 2: Stage 1: Resource Collection (best viewed on a screen)

Fig. 3: Stage 2: Input Generation

There are several existing data sets containing LiDAR tests consisting of PC gathered from the real world with accompanying $ExpSem$ from human annotation for general purpose scenes (i.e., [21]) and for vehicle-centric scenes (i.e., [2], [6], [9], [32], [35], [39]). In addition to the PC labeling, many data sets (i.e., [2], [6], [32], [39]) also provide a list of *entities*, E, describing at a semantic level how groups of points form a logical group in the real world, e.g. the set of points corresponding to a single vehicle. Although such data sets are expensive to collect and annotate, this is a one time fixed cost which allows our approach to greatly extend the utility of these data sets without incurring additional data collection costs.

B. Resource Collection

The first stage of our approach leverages the provided data set to generate a resource set for the later mutation operations. Figure 2 shows the full data pipeline. Our technique performs the Resource Collection stage in two phases.

First, the Entity Identification phase (Fig. 2b) attempts to find additional entities that were not initially recorded in the data set. This is useful due to the different entity identification paradigms used to generate existing real-world data sets in order to provide a rich and consistent entity set. For example, the SemanticKITTI data set [2] does not include traffic signs in its entity lists; however, it does label points with a traffic sign class. Thus, a traffic sign entity can be reconstructed by identifying a collection of neighboring labeled traffic sign points. For each of the provided $(PC, ExpSem, E_{orig})$ in the data set, the semantic information of $ExpSem$ is used along with the spatial relationships in PC and a set of parameterized invariants (further described in Section III-A) to identify additional entities that were not provided with the initial entity list, E_{add}, resulting in an expanded $E_{total} = E_{orig} \cup E_{add}$.

Second, each E_{total} is filtered to identify entities suitable for mutation in generating test cases: $E_{filtered} = Filter(E_{total})$. Entities that are too small, too far from the sensor, or partially o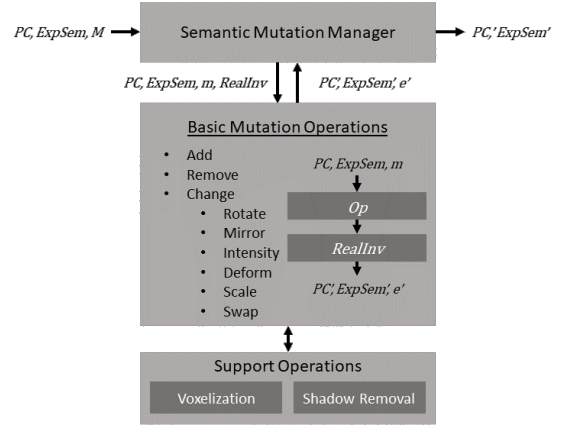bscured are removed. This process is parameterized and helps ensure that the later stages of test case generation provide realistic test inputs. Removing entities that are too small or too far from the sensor helps to ensure that the entities have sufficient definition and are not affected by sensor noise. Removing entities that are partially obscured ensures that the points that form the entity are complete (e.g. the car is not missing its hood because it was obscured by another car). Once complete, the Resource Collection stage outputs a resource set consisting of a set of $(PC, ExpSem, E_{filtered})$ that are suitable for mutation. This process only needs to be completed once for a given data set.

C. Mutation Generation

The second stage of the technique focuses on generating a realistic mutation, Mut, that takes as input a PC, PC, its expected semantics, $ExpSem$, and a parameter set, M, and produces a novel input and expected semantics:

$$Mut(PC, ExpSem, M) = (PC', ExpSem') \qquad (1)$$

where the parameter set M consists of what type of mutation should be performed along with any mutation-specific parameters, such as what entities from the resource set should be used, and a set of realism invariants $RealInv$ that must hold during the mutation process in order for the resulting PC to be realistic. During the mutation process, if an invariant is violated, the mutation will abort and a new test will not be generated; this means that the provided $PC, ExpSem, M$ were incompatible. We adopt the definition of realism proposed by [47], that a test case is realistic if $PC, ExpSem$ could come from a possible configuration of the real world, including the long tail of improbable configurations.

This stage is broken into several components as outlined in Figure 3. First, the Semantic Mutation Manager breaks the mutation M into a composition of basic mutation operations at the entity level, each with their own parameters m. Each of these operations falls into one of three categories: add, remove, or change, and produces intermediate results for the Semantic Mutation Manager by either updating PC and $ExpSem$, or by altering an entity, e', to be used in the next operation, e.g.

Mirror first produces a mirrored e' that Add then inserts into PC and $ExpSem$. The basic mutations are further supported by a set of common operations that are used to generate the intermediate PC' and $ExpSem'$ and enforce the invariants.

1) Basic Mutation Operations: Each of the basic mutation operations are based around a single entity and emulate a physical change that could occur in the real world to ensure realism. Each operation has specific invariants that we have recognized as key concerns for that operation which are briefly introduced here, with further discussion on parameterization in Section III. Each mutation is performed on both PC and $ExpSem$ simultaneously so that the mutated $ExpSem$ remains a valid semantic interpretation for the mutated PC. Table I introduces the eight basic mutation operations we explore. While these provide a rich baseline, the "change" category could be further expanded based on desired testing directions as each new operation provides a realistic mechanism to further expand the diversity of the generated tests in a new dimension.

2) Common Operations: In this section, we introduce two operations common to multiple basic mutation operations.

Voxelization: Many invariants, e.g. those of the Rotate operation, rely on checking that an entity's location is allowable under a given constraint, which can be expensive to validate. Voxelization is a technique rooted in computer graphics [10], [13] that discretizes a 3D representation into an occupancy grid, allowing for efficient computation of functions such as entity intersection or proximity in invariant verification.

Shadow Removal: LiDAR sensors fundamentally work by casting rays of light out at regular intervals and using the reflected beam to determine the existence of other objects. This means that each object detected by the LiDAR casts a "shadow" as it occludes all objects behind it. When a LiDAR PC is mutated, for example with the Add operation, it is possible for altered points to now occlude existing points, violating this invariant. To restore this property and ensure the invariants are respected, we use a common shadow removal subroutine. The shadow is removed by taking the convex hull of the entity and projecting a mask from the LiDAR sensor to the edges of the entity and then removing any points that fall within that polygon, restoring the shadow.

At the end of stage 2, our approach has produced a valid mutation that consumes a given PC, expected interpretation, parameter set, and set of invariants to produce a new PC and expected interpretation: $Mut(PC, ExpSem, M, RealInv) = (PC', ExpSem')$. Next, in stage 3, the approach will use this information to build the oracle to evaluate the peformance of an SUT on this new test, $eval$.

D. Performance Evaluation

Existing approaches for evaluating performance [20], [28], [47] suffer from two key limitations: inability to identify perception failures affecting a small but critical portion of the input, e.g. not identifying a pedestrian, and inability to provide meaningful results if large portions of the test case are mutated or if the SUT performed poorly on the original
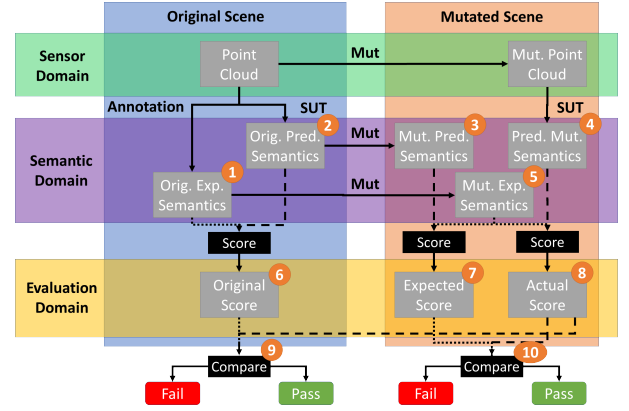

Fig. 4: Stage 3: Performance Evaluation

test case. We briefly explore these limitations and demonstrate how our approach resolves these issues.

Let $t = (PC, ExpSem)$ be an initial test case that is mutated to form a new test case $t' = (PC', ExpSem') = Mut(PC, ExpSem, M, RealInv)$. For brevity, we write $t' = Mut(t)$, etc. In prior literature [20], [28], [47], the score of the SUT on a mutated test case is based on the intersection over union (IoU) metric when comparing the SUT's PC prediction with the expected semantics, i.e. $score(SUT(PC'), ExpSem')$. When IoU is calculated over the whole PC, analogous to calculating IoU over the whole image as in prior literature [47], this is a measure of accuracy in terms of the percentage of points correct; later referred to as the "Accuracy metric". This was also used in past work as the evaluating oracle; if the SUT's IoU score of the mutated test was ϵ below the SUT's IoU score of the original test, then the SUT failed the test[2]:

$$\overset{⑥}{origScore} = score(\overset{①}{\overbrace{ExpSem}}, \overset{②}{\overbrace{SUT(PC)}})$$

$$\overset{⑧}{mutScore} = score(\overset{⑤}{\overbrace{Mut(ExpSem)}}, \overset{④}{\overbrace{SUT(Mut(PC))}})$$

$$\overset{⑨}{\epsilon} > origScore - mutScore \qquad (2)$$
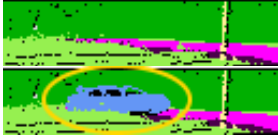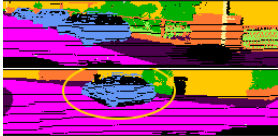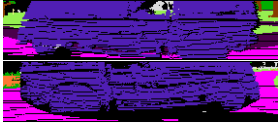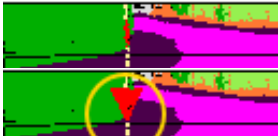
Additionally, though not used as an oracle, if IoU is calculated over only a subset of points, e.g. points belonging to a certain class as in prior LiDAR evaluation [20], [28], then this metric varies as the number of points in the class changes. This means that in the face of mutations that alter the number of points in a class, the difference evaluation above is not meaningful.

Solely relying on an Accuracy metric is ill-suited to LiDAR-based perception systems due to the nature of the entities of interest and possible failure modes; in particular entities "disappearing" or being entirely mislabeled. For example, if the SUT mislabels a person in a scene the person would represent $\approx 0.05\%$ of the total points in the scene producing a minuscule drop in accuracy in spite of its criticality. This

---

[2]The order matters in Equation 2; if $\epsilon > mutScore - origScore$, then the mutation improved performance and is not problematic.

TABLE I: Basic Mutation Operations

Name	Category	Motivation	Parameters	Invariants	Example (Before, After)
Add	Add	A crucial limitation of available real-world LiDAR test data is the large imbalance in the semantic classes; many of the most safety critical classes, e.g. pedestrians, cyclists, etc. are scarce and can be increased through this mutation.	• Entity to add	Entity must appear on the ground, and not be occluded by or intersect any other entities. Entity must be placed at the same location in the new PC as it was in its original PC	
Remove	Remove	SUTs may use the presence of certain entities as context clues when labeling other aspects of the scene, so removing an entity may change the SUTs performance. Once the entity is removed, suitable background points must be added to the PC to replace where the entity used to exist.	• Entity to remove	The entity to remove must not obscure another entity. Filled in points must be of an appropriate semantic class for the location removed.	
Rotate	Change	Due to the rotational nature of the LiDAR sensor, the location of the entity can rotate around the sensor while maintaining the correct perspective. This movement enables an entity to appear in a context that it did not appear in the data set, e.g. a pedestrian in the roadway.	• Entity to rotate • Angle of new location	Once in the new position, the entity must not be occluded by scenery.	
Mirror	Change	Many common entities in LiDAR scenes can be viewed from a mirrored orientation, e.g. the reverse side of a car. Mirroring increases diversity and rare scenes, such as a vehicle facing the wrong direction on the street. Although likely not present in any training data, it is imperative that a system respond appropriately to this situation.	• Entity to mirror	No additional invariants; the mirrored entity still subject to Add invariants.	
Adjust Intensity	Change	LiDAR sensors also detect the intensity of the reflected beam from a surface which can be affected by the material, reflectivity, etc. of the surface. For example, different vehicles naturally have different intensity profiles based on their different paint characteristics and thus this operation is analogous to repainting a car.	• Entity to adjust intensity • Amount to adjust intensity	Intensity is not adjusted in certain areas, i.e. areas that could not be repainted such as a license plate.	
Deform	Change	Although most vehicles of a given make, model, and year share the same shape, vehicles may be involved in collisions or otherwise accumulate deformations to their exterior; similarly, traffic signs share a common reference shape, but may wear or deform over time. To emulate this, we mutate the entity by deforming a portion of the surface, resembling a dent.	• Entity to deform • Location to deform • Size of deformation	Entity must be deformable, e.g. a vehicle but not a person, and the deformation must be of appropriate size for the entity	
Adjust Scale	Change	The scale of a class of entity can to change over time and across locations. For example, the SemanticKITTI data set [2] was collected in 2012, and in the 10 years since, vehicles have gotten larger. Thus, an entity that is observed in the data set may be similar to an entity that could be encountered in the future, only slightly larger or smaller.	• Entity to scale • Amount to scale	Entity to scale must contain enough points to have sufficient definition to increase scale; amount to scale must be appropriate for the entity	
Class-Preserving Swap	Change	The original locations of the entities serve as a blueprint for where an entity of a given class could be found in the real world, allowing for the replacement of that entity within its class. For example, the specific traffic sign at a given location could be exchanged for another, e.g. a stop sign could be replaced with a yield sign or a speed limit sign with a warning sign. While this does not affect the expected interpretation, this mutation can be used in robustness testing to bring additional variety to the test suite.	• Entity to replace • Detail of replacement	The replacement must be of similar size to the original entity.	

5

implies that a small ϵ is required to identify the safety-critical issues that we are interested in finding, yet this does not differentiate what causes the drop in accuracy and could lead to many nuisance failures, e.g. when the SUT mislabels 0.05% of PC as "terrain" instead of "vegetation".

This issue is further exacerbated if the mutation changes large portions of the input. Consider a mutation that removes a car. Under Equation 2, if the SUT originally mislabeled the car, it would have a low $origScore$. Then, if the mutation replaced the car with entities the SUT labeled correctly, its $mutScore$ would improve by the amount of the car. However, if in the mutated case the SUT also mislabeled a pedestrian, its $mutScore$ would still likely be an improvement over its $origScore$ since a pedestrian is generally smaller than a car.

To address this, we introduce two changes to judge the output in the LiDAR domain: relative success metrics and Jaccard Index.

1) Relative Success Metrics: To address the issue of the mutation changing the test case, we adopt a relative success metric, where an SUT's performance on $t' = Mut(t)$ is compared to its *expected* performance based on its performance on t. Under the scheme in Equation 2, the SUT's performance is evaluated based solely on its separate performance on t ($origScore$) and t' ($mutScore$). However, under the new scheme, we introduce a new term that performs the mutation operation on the SUT's original prediction for t.

$$\overset{\text{⑦}}{expScore} = score(\overset{\text{⑤}}{\overbrace{Mut(ExpSem)}}, \overset{\text{③}}{\overbrace{Mut(SUT(PC))}})$$

$$\overset{\text{⑧}}{mutScore} = score(\overset{\text{⑤}}{\overbrace{Mut(ExpSem)}}, \overset{\text{④}}{\overbrace{SUT(Mut(PC))}})$$

$$\overset{\text{⑩}}{\epsilon} > expScore - mutScore \tag{3}$$

Figure 4 visualizes the symmetry of this approach, highlighting how this provides a more direct comparison.

2) Jaccard Index: As discussed above, prior work used IoU as the scoring metric for evaluating performance. However, the scoring function need not be fixed. The Jaccard Index is a metric commonly used in LiDAR classification evaluation due to its ability to provide a more nuanced metric based on the class label of the affected object [2], [11]. This is particularly useful in the LiDAR context due to the extreme class imbalance in the data; for example, in the SemanticKITTI data set there are roughly 4 orders of magnitude more points labeled "road" than "motorcyclist". Jaccard Index is calculated by the formula:

$$\frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c + FN_c} \tag{4}$$

Where C is the number of classes, the summation iterates over each of the classes c, and TP_c, FP_c, and FN_c are the number of true positive, false positive, and false negative points respectively for the class. Each class contributes equally to the score, allowing for changes in a relatively small class to be reflected in the overall score. Returning to the example

of the SUT missing a person, if an SUT mislabels one out of three people in a scene, this is only a $\approx 0.05\%$ drop in overall IoU accuracy; however this would lead to a 33% drop for the "person" class, which out of 20 classes would be a 1.5% drop in the Jaccard Index. This means that for under-represented classes, the Jaccard Index provides a better aggregate score. Together, both accuracy and Jaccard Index provide important metrics about the SUT's performance on the test.

III. Implementation

We developed an extensible implementation of the approach in Python that automatically generates novel LiDAR test cases through mutation. Each mutation is implemented using Numpy [23], SemanticKITTI's developer tools [2], and open3D [53]. In this section, due to space constraints, we only describe the implementation of the Resource Set Builder and three representative mutation operations; a detailed description of all operations is available in the repository.

A. Building the Resource Set

When building the resource set, the filters to identify entities that are too small, too far from the sensor, or occluded are parameterized as follows. First, the entity must contain more than $minPointThreshold$ points and the distance to the entity must be less than $distThresh$. Second, to filter occluded objects, the bounding box of the entity is acquired and scaled based on the distance from the sensor by a factor of at most $occlBuffer$ to provide a buffer to ensure all occluding objects are identified. The box is also extended to the ground in case another entity is occluding the bottom portion of the object, for example, a car that is only comprised of the roof points. A frustum (bounding cone) is drawn from the sensor to the altered bounding box and if any points that are not semantically classified as unlabeled, outlier, or some kind of ground (road, sidewalk, etc) are included within this new polygon the entity is discarded. As described in Section III-A, the component also performs Entity Identification to automatically identify entities not completely captured in the data set. Because SemanticKITTI does not capture sign entities, the implementation only seeks to identify sign entities. To accomplish this, the points labelled as poles and signs are isolated and grouped into clusters using open3D's DBSCAN (Density-Based Spatial Clustering of Applications with Noise). If a cluster has both sign points and pole points, has more than $minSignPoints$ points, is smaller than $signSizeThreshold$, and meets the other resource set criteria, it is classified as a unique sign entity and added to the entity list.

B. Mutations

Add Entity. An entity is added into a scene at a given location by adding its points at that location and then removing its shadow. To ensure that the entity appears with the correct size, the distance from the LiDAR sensor is kept consistent. When adding an entity, $RealInv$ verifies that the entity does not intersect another entity by voxelizing the non-ground points and comparing this to the entity's new location. The entity is

checked for occlusion by drawing a frustum from the entity's bounding box to the sensor and ensuring that there are fewer than $addOcclusionThreshold$ points in the frustum. Finally, $RealInv$ verifies that the entity is being placed on the ground in a location that is semantically viable. To do so, the viable ground points are flattened and voxelized and a portion of the entity's points greater than the $groundThreshold$ must be above the ground voxels. Due to its point of view, the ground may not be visible within a short distance of the sensor. To accommodate, if the entity is closer than the $groundDistanceThreshold$, this check is ignored.

Remove Entity. To remove an entity from a scene, the shadow removal process is run in reverse. First, the entity's shadow is generated, showing the area that must be filled to replace the entity. The shadow area is then split in two halves vertically and each half is swept rotationally outward to identify an equal sized portion adjacent to the entity that could be used to back-fill the vacant area. These points are copied into the area, preserving the distance of the points and semantics surrounding the missing area. During this process, $RealInv$ verifies that the entity to be removed does not partially obscure another entity and that the points that will fill in the empty area have appropriate semantics. To avoid obscuring another entity, there must be fewer than $removePointsAboveThresh$ points above the entity. To ensure appropriate semantics of the filled area, the filled points must not contain points labeled in the $invalidReplacementClasses$ set. These classes are chosen since they are likely to result in an unrealistic replacement; e.g., if there are building points in the replacement, these points will be disconnected from the rest of the building.

Rotate Entity. Used in conjunction with the Add Entity mutation, this mutation is used to identify a location within a scene where an entity could be placed and align the entity with that location, while $RealInv$ ensures that it is not occluded by scenery. To do so, a line is drawn from the sensor to the entity's center at the rotated position. The non-road points are voxelized and if the line intersects with any of these voxels, the mutation is not performed. If valid, the entity is aligned with the road at that position.

IV. EVALUATION

To evaluate our test generation technique for LiDAR perception systems, we pose the following research questions:

- **RQ1.** How effective is this technique in uncovering failures defined at different levels of severity and which mutations are the most effective?
- **RQ2.** How effective is this technique at generating realistic test cases?
- **RQ3.** How efficient is this technique in terms of the time to generate the mutations?

A. SUTs

We evaluate our technique through testing five highly competitive perception systems submitted to the SemanticKITTI benchmark for the "Semantic Segmentation" task [2], shown in Table II. We evaluate against these models as they were the top

TABLE II: SUT Rankings

Rank		Name	Year	mIoU
Overall	With Code			
3	1	Cylinder3D [55]	2020	67.8
4	2	SPVNAS [40]	2020	66.4
5	3	JS3C-Net [?]	2021	59.5
11	5	SalsaNext [11]	2020	59.5
15	7	SqueezeSegV3 [48]	2021	55.9

performing models that included code and a pre-trained model. Each model's code was forked, edited to produce a consistent output format, and containerized in Docker for reproducability.

B. Parameterization for Test Generation

We used the SemanticKITTI [2] data set as the backing data set for our mutations. As described in Section III-A, the initial Resource Collection stage identifies additional entities and then filters out entities not suitable for mutation. During this phase, the $259,347$ entities in the data set and $8,443$ additional entities identified were filtered to a set of $63,323$ entities suitable for mutation[3]. We explore seven distinct compositions of the basic mutation operations, described in Table III, so selected to showcase the ability of the operations to produce a diverse set of test cases. These compositions and parameterizations are not exhaustive; e.g. the add operation could be combined with the scale, deform, or adjust intensity operations to produce more variety in the added entities. Future study should examine directions for expansion that will provide meaningful utility in the types of tests generated.

TABLE III: Mutation Compositions Studied

Composition	Entity Classes Considered
Add ∘ Rotate	Any
Add ∘ Mirror ∘ Rotate	Any
Remove	Any
Adjust Intensity	Vehicle
Deform	Vehicle
Scale	Vehicle
Class-Preserving Swap	Sign

We generated $10,000$ test cases for each of the 7 composite mutations. Full details of the values selected for the parameters described in Section III are available in the repository. The values selected represent reasonable defaults based on small-batch experimentation during tool design. For each test case, except for Remove mutations, the entity and mutation parameters, such as the location to deform or amount to adjust intensity, were chosen at random. Removal's stringent invariants to ensure realism made it difficult for the technique to find suitable replacement points in PC, leading us to enumerate all $6,852$ possible mutations instead of using random selection.

C. Results

1) RQ1 Results: Failures Found at Different Thresholds: As discussed in Section II-D, we investigate both the Accuracy

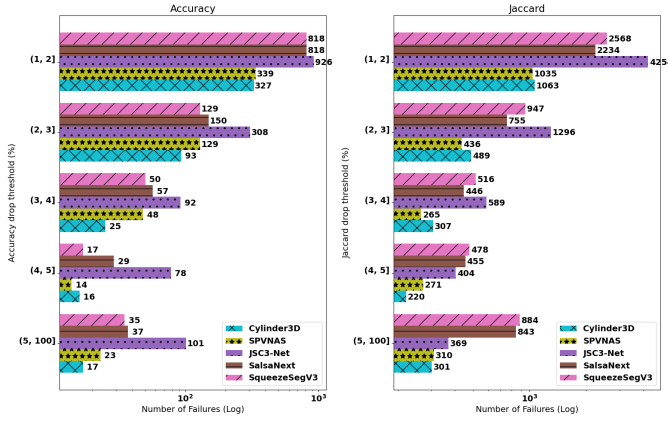[3]A breakdown per class is available in our GitHub repository

Fig. 5: Failure Counts at Different Thresholds

Fig. 6: SUT Failures by Mutation at Highest Severity

(a) JS3C-Net Prediction Before Mutation

(b) JS3C-Net Prediction After Mutation

(c) Expected Semantics After Mutation

Fig. 7: JS3C-Net Performance on Vehicle Scale Mutation
Prediction change in red oval, mutation change in yellow oval

and Jaccard metrics to assess a SUT's performance. Since the evaluation metric (Eq. 3) proposed uses a tunable threshold, ϵ, to determine if an SUT passed or failed, we examine the effects of this parameter, grouping the failures observed by their drop in score; larger values of ϵ correspond to more severe failures. We consider ϵ values $\{(1, 2], (2, 3], (3, 4], (4, 5], (5, 100]\}$. Under the Accuracy metric, $\epsilon = 5$ corresponds to 5% of the PC being misclassified, indicating large portions of the scene are incorrect. Under the Jaccard metric, since there are 19 classes evaluated, this roughly corresponds to one class being entirely misclassified ($1/19 \approx 5.3\%$).

Fig. 5 shows the number of failures per SUT for each mutation type on the x-axis in log scale and the failure severity (as per the ϵ) on the y-axis; the left and right plots show the Accuracy and Jaccard metrics respectively. This shows that the tests generated by the technique identified between 17 and 101 failures among the 5 SUTs at the highest severity level ($\epsilon = 5$) under the Accuracy metric and at least 301 under the Jaccard metric. Breaking down the performance by SUT, we also see that JS3C-Net is considerably outperformed by the other SUTs, including those ranked lower in the original leaderboard shown in Table II.

To investigate further, we examine the breakdown of SUT performance based on the mutation type. Fig. 6 shows the number of failures at the highest level of severity ($\epsilon = 5$) per mutation. Every mutation yielded at least one failure except for the Sign Replace mutation under the Accuracy metric. As expected, the Jaccard metric identifies more failures, and more failures at higher thresholds, than the Accuracy metric since it is more sensitive to failures affecting small classes. Other than the Add Rotate and Add Mirror Rotate mutations, which yield similar failure results for all SUTs, the different SUTs are susceptible to different mutations. For example, JS3C-Net appears to be particularly brittle to Vehicle Deform and Vehicle Scale and to a lesser extent Remove and Vehicle Intensity.

We illustrate the type of failures found through Fig. 7 (others are provided in the repository), which shows a Vehicle Scale test case that uncovered a high-severity failure in JS3C-Net with an Accuracy drop of 21.9%. Fig. 7a shows JS3C-Net's
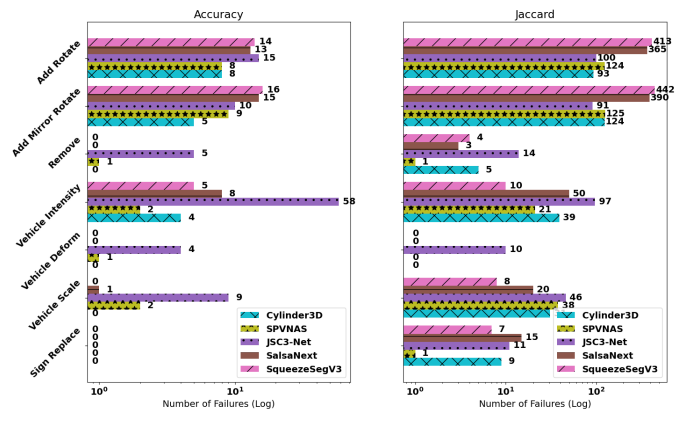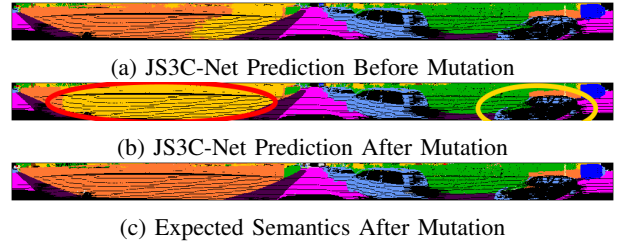
prediction on the unedited PC. The mutation then adjusts the scale of the vehicle on the right of the PC, and Fig. 7b shows the prediction on the mutated PC. Although the performance around the altered vehicle changes slightly, the accuracy drastically changes due to the large change in classification on the left side of the PC. In these images, orange corresponds to "fence" and yellow to "building"; as seen in Fig. 7c, the expected classification is fence for the entire area. While JS3C-Net classified more as fence than building before the mutation, after the mutation this reversed to more classified as building than fence, leading to a large decrease in accuracy. Such findings point to directions to improve robustness.

Fig. 8 illustrates another failure, this time in Cylinder3D identified by a drop in the Jaccard metric of 7.6%. This mutation added a bicyclist down the road from the LiDAR. Fig. 8a shows Cylinder3D's actual prediction for this test case while Fig. 8b shows the mutated predicted semantics, i.e. the expected output for Cylinder3D on the mutated input. Since in the SemanticKITTI coloring scheme the coloring for "bicyclist" is very similar to "road", we have added a white outline in the picture to facilitate its visualization. Note that in Fig. 8a the bicyclist is composed of two regions, one cyan and one red corresponding to "bicycle" and "person" respectively. Thus, while Cylinder3D did identify the components of the "bicyclist" correctly, it did not identify them as a cohesive unit. Subtle differences such as this can be very important for perception and any components that rely on perception

(a) Cylinder3D Prediction After Mutation

(b) Cylinder3D Mutated Predicted Semantics (Expected Score)

Fig. 8: Cylinder3D Performance on Add Rotate Mutation Prediction change in red oval

TABLE IV: False Positive Counts, V=3

Mutation	Accuracy		Jaccard	
	TP	FP	TP	FP
Add Rotate	39	2	634	72
Add Mirror Rotate	42	2	672	73
Scene Remove	6	—	26	—
Vehicle Intensity	63	2	178	4
Vehicle Deform	5	—	10	—
Vehicle Scale	12	—	81	14
Sign Replace	—	—	37	1
Total	167	6	1638	164
%	(97%)	(3%)	(91%)	(9%)

data. Under the SemanticKITTI labeling instructions, bicycles *cannot* be mobile while bicyclists can; this difference could be imperative to planning a safe trajectory for the system. While this difference is a 0.03% drop in the Accuracy metric, it is a 7.6% drop in the Jaccard metric due to the change in classes—"bicyclist" to "bicycle" and "person"—which Jaccard is uniquely suited to identify.

2) RQ2 Results: False Positives: When evaluating the performance of an SUT on a given test case, we want to ensure that the test case is realistic, as a failing but unrealistic test case is a false positive. The invariants described in Section II-C aim to prevent such false positives; however, these invariants must balance the need for realism with the ability to generate a diverse test set and in practice some false positives will occur. Determining if a given test case is realistic is by definition a difficult problem—any method for determining realism could be used to improve the performance of the perception systems as doing so would allow for a complete characterization of the long-tail distribution of the real-world input space. Prior work on semantic mutations for images relied on human intuition to judge if a camera image was realistic [47]. However, human intuition is not well suited to judging conformity of LiDAR PCs, as the sensor modality is far removed from how we perceive the world[4]. Other related work on LiDAR input generation did not evaluate realism for individual test cases, instead using aggregate performance numbers to show similar

[4]Examples of false positive PCs as determined by our technique and further discussion on human detection are available in our GitHub repository.

results under real and generated test cases to argue that all test cases should be assumed to be realistic [14], [28]. To address these limitations, we introduce a voting mechanism to determine if a generated input for a LiDAR perception system is realistic. Given a set of SUTs, S, trained on real world data, and a test case t', we evaluate each $s \in S$ on t'. We then use the count of SUTs that fail on t', v, to judge realism based on a voting threshold V. If $v < V$ SUTs fail, then we determine that the failure is not a false positive and can be attributed to a fault in the SUT. If $v \geq V$ SUTs fail, then we determine that the failure is a false positive due to unrealistic test input. This method allows for *automated* false positive detection, but requires a diverse set of well performing SUTs. Diversity is required as a realistic test case that results in failures for multiple systems will be judged as a false positive which can lead to an overapproximation of false positives.

Table IV shows the number of false positive test cases for each of the mutation types. This includes only the highest level of failure examined in Section IV-C1 ($\epsilon = 5$). Selecting $V = 3$, corresponding to the majority of SUTs, we find only 3% and 9% of tests are deemed to be false positives under the Accuracy and Jaccard metrics respectively; if we impose the strictest criteria of $V = 2$, the rate increases to 19% and 36%. With the false positivity rate affected by both the choice of invariant parameters and V, this rate indicates a reasonable balance of finding impactful test cases while not spending too much execution time on false positive tests.

3) RQ3 Results: Efficiency: We now investigate the efficiency of the technique in terms of the time taken to generate the mutations. The average times per mutations were: $1.90s$ for Add Rotate, $1.97s$ for Add Mirror Rotate, $9.62s$ for Remove, $0.46s$ for Vehicle Intensity, $0.43s$ for Vehicle Deform, $0.84s$ for Vehicle Scale, and $0.71s$ for Sign Replace. The high time per test for Remove is due to the stringent invariant requirements. Overall, these mutations are generally slower than the majority of the SUTs' time to label a scene—on average Cylinder3D took $0.59s$, SPVNAS took $0.17s$, JS3C-Net took $4.23s$, SalsaNext took $0.03s$, and SqueezeSegV3 took $0.16s$. However, the technique test generation times are within one order of magnitude of test execution times. This is encouraging for several reasons. First, as a test generation technique, the tests created can be reused for future tests without incurring the full cost of generation, mitigating a high up-front cost. Second, although the costs of generation may be higher than the costs of evaluation, they are still pragmatic. Third, as the time taken to generate a mutation increases with the strictness of the invariants imposed, the timing is indirectly a tunable parameter and could be altered based on testing goals. Finally, as a research prototype, general efficiency refinements to the tool implementation have the opportunity to decrease the time per mutation.

D. Threats to Validity

The external validity of our findings are affected by our choice of data set and SUTs. We selected the SemanticKITTI data set for its popularity as a benchmark for LiDAR-based

9

perception systems, and we selected the top five performing SUTs that were publicly available. Extending to additional available benchmarks and SUTs would help to generalize the findings. The internal validity of our findings may be affected by the implementation of the mutation pipeline which is complex and contains external components and many parameters. To mitigate these issues, we publicly release our code and provide documentation of the parameters used in the study, as well as all raw study results. Additionally, while the parameters chosen represent reasonable default values determined during system design, individual testing goals or testing using different data sets may necessitate changes to these values. In terms of construct validity, though we present a novel method for automatically assessing PC realism, the appropriate parameterization of this approach and other methods for external validation remain an ongoing challenge.

V. RELATED WORK

In this section, we explore the current state of testing LiDAR-based perception systems, focusing on the current weaknesses and differences from our approach.

A. Reality and Simulation

Real world testing is the gold standard for validation of ASs as it reflects the operating conditions the system will encounter when deployed; however, its effectiveness is constrained due to limited diversity. Although it is straight-forward to collect typical data in the real world by operating the system, it is impractical or dangerous to collect the safety-critical scenarios required for testing [27]. Further, labeling collected data with ground truth interpretation comes at great expense; the SemanticKITTI data set required over 1700 hours of human effort to annotate [2], and other data sets have noted the need for specialized training and software with multiple stages of review [39]. Combined, these factors limit the availability of diverse test inputs gathered from the real world.

To improve diversity, testing using simulation is widely used as new unseen scenarios for testing can generated relatively cheaply [31], [37], [42]. Existing approaches have used general purpose autonomous driving simulators [12], LiDAR-specific simulators [19], or adapted game engines [44], [52]. Prior work has noted that the major limitation of the simulation testing approach, the simulation-reality gap [25], remains a relevant issue in the LiDAR testing domain [44]. Recent work has attempted to improve realism by supplementing simulated data with real-world data [28], [46]; however, the simulation-reality gap remains a significant challenge.

B. Input Generation

1) Weather Simulation: LiDAR perception is affected by weather which can increase noise and occlude points [36]. Prior work has explored generating weather effects including snow, fog, and rain [20], [22], [26]. While validating weather performance is crucial to safety, the expected output in these weather-based test cases is the same as the expected output before weather effects are introduced, limiting diversity.

2) Adversarial Testing: Machine learned components are known to exhibit unstable behavior where a slight change in the input can lead to a drastic change in the output [18] and LiDAR-based perception systems are no exception [54]. This leads to adversarial attacks which try to alter the prediction in a specific way through small changes to the input, with the potential to produce system-level failures [7], [43], [56]. While improving robustness is important to AS safety, the niche inputs do not improve general test suite diversity.

3) Data Augmentation: Data augmentation techniques take as input a prior input and expected output pair and generate a novel input and expected output pair. Several techniques for images have been extended to LiDAR PCs; for example, affine transformations [20], or fusing together different scenes [24]. Due to this global effect, the semantics of each point is the same before and after even if the point is moved, limiting improvements to diversity. Prior entity-specific augmentations have removed points, added noise, or mixed together two entities [8], though without considerations for realism. Another technique for augmentation randomly took points from other scenes and added them to the input scene [38], [49], [51]. However, the randomness could lead to physically infeasible point locations; following techniques improved this by using 3D models and simulating how the object's addition would affect the point locations [14]. Although this improves diversity, the lack of consideration for PC intensity may affect realism.

VI. CONCLUSION

LiDAR inputs are critical to modern perception systems, but mechanisms to generate valid inputs and their expected outputs are lacking in realism and diversity to match the safety-critical nature of these systems. We introduce a novel approach to semantic mutation testing of LiDAR-based perception systems that creates realistic and diverse test cases by leveraging existing point clouds as both a base for mutation and a resource for enabling other mutations, constructing mutations that are designed to address weaknesses in existing data sets, and deriving richer oracles that identify potentially safety-critical issues in perception performance. Further, our implementation and its usage as part of a study demonstrates the ability of the approach to efficiently find potentially safety-critical failures in LiDAR-based perception systems.

This sets several directions for future work. ASs use LiDAR in conjunction with other sensors simultaneously and may integrate multiple sensors for perception; future work should explore mixed-input generation that mutates data across multiple sensor modalities simultaneously. Further, ASs operate in continuous time and may use prior perception knowledge when perceiving the current environment; future work should explore mutations over sequences of point clouds and explore the richer mutation space available across multiple time steps.

VII. DATA AVAILABILITY

The data and artifacts, much of which we could not include in the paper, are available at https://github.com/anonymoussubmission11936/Anon11936

REFERENCES

[1] Evan Ackerman. Ford and baidu invest $150 million in velodyne for affordable lidar for self-driving cars. *IEEE Spectrum*, Aug 2016.

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[3] Rebecca Bellan. Motional launches autonomous hyundai ioniq 5s on lyft network in las vegas. *TechCrunch*, Aug 2022.

[4] NTS Board. Collision between vehicle controlled by developmental automated driving system and pedestrian. nat. transpot. saf. board, washington, dc. Technical report, USA, Tech. Rep. HAR-19-03, 2019. URL https://www. ntsb. gov/investigations . . . , 2019.

[5] Neal E Boudette and Niraj Chokshi. U.s. will investigate tesla's autopilot system over crashes with emergency vehicles. *New York Times*, Aug 2021.

[6] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[7] Yulong Cao, Chaowei Xiao, Dawei Yang, Jing Fang, Ruigang Yang, Mingyan Liu, and Bo Li. Adversarial objects against lidar-based autonomous driving systems. *CoRR*, abs/1907.05418, 2019.

[8] Jaeseok Choi, Yeji Song, and Nojun Kwak. Part-aware data augmentation for 3d object detection in point cloud. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3391–3397, 2021.

[9] Yukyung Choi, Namil Kim, Soonmin Hwang, Kibaek Park, Jae Shin Yoon, Kyounghwan An, and In So Kweon. Kaist multi-spectral day/night data set for autonomous and assisted driving. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):934–948, 2018.

[10] Daniel Cohen-Or and Arie Kaufman. Fundamentals of surface voxelization. *Graphical models and image processing*, 57(6):453–461, 1995.

[11] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. Salsanext: Fast semantic segmentation of lidar point clouds for autonomous driving. *CoRR*, abs/2003.03653, 2020.

[12] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[13] Elmar Eisemann and Xavier Décoret. Fast scene voxelization and applications. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 71–78, 2006.

[14] Jin Fang, Xinxin Zuo, Dingfu Zhou, Shengze Jin, Sen Wang, and Liangjun Zhang. Lidar-aug: A general rendering-based augmentation framework for 3d object detection. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4708–4718, 2021.

[15] Ryan Felton. Lidar maker velodyne shifts away blame in fatal uber self-driving crash, Mar 2018.

[16] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, and Qi Alfred Chen. A comprehensive study of autonomous vehicle bugs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 385–396, 2020.

[17] Richard Gonzales. Feds say self-driving uber suv did not recognize jaywalking pedestrian in fatal crash, Nov 2019.

[18] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[19] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, pages 199–208, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[20] An Guo, Yang Feng, and Zhenyu Chen. Lirtest: Augmenting lidar point clouds for automated testing of autonomous driving systems. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2022, page 480–492, New York, NY, USA, 2022. Association for Computing Machinery.

[21] T Hackel, N Savinov, L Ladicky, JD Wegner, K Schindler, and M Pollefeys. Semantic3d. net: a new large-scale point cloud classification benchmark. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 41:91–98, 2017.

[22] Martin Hahner, Christos Sakaridis, Mario Bijelic, Felix Heide, Fisher Yu, Dengxin Dai, and Luc Van Gool. Lidar snowfall simulation for robust 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16364–16374, June 2022.

[23] 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. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[24] Frederik Hasecke, Martin Alsfasser, and Anton Kummert. What can be seen is what you get: Structure aware point cloud augmentation. 2022.

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

[26] Velat Kilic, Deepti Hegde, Vishwanath Sindagi, A. Brinton Cooper, Mark A. Foster, and Vishal M. Patel. Lidar light scattering augmentation (lisa): Physics-based simulation of adverse weather conditions for 3d object detection, 2021.

[27] Waymo LLC. Waymo's safety methodologies and safety readiness determinations. Technical report, October 2020.

[28] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world, 2020.

[29] Post author By Luminar Marketing. Luminar and volvo cars announce plans for next generation suv to be revealed in 2022, Jan 2022.

[30] Aarian Marshall. Uber video shows the kind of crash self-driving cars are made to avoid, Mar 2018.

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

[32] Abhishek Patil, Srikanth Malla, Haiming Gang, and Yi-Ting Chen. The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9552–9557. IEEE, 2019.

[33] Jake Port. From the vault: Lidar: How self-driving cars 'see' where they're going, Jan 2022.

[34] Sabbir Rangwala. Amazon's zoox acquisition– is lidar next? *Forbes*, Aug 2020.

[35] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, 2018.

[36] Fredrik Schalling, Sebastian Ljungberg, and Naveen Mohan. Benchmarking lidar sensors for development and evaluation of automotive perception. In *2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–6, 2019.

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

[38] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *CoRR*, abs/1812.04244, 2018.

[39] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

[40] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. *CoRR*, abs/2007.16100, 2020.

[41] Brad Templeton. Tesla in taiwan crashes directly into overturned truck, ignores pedestrian, with autopilot on. *Forbes*, Jun 2020.

[42] Christopher Steven Timperley, Afsoon Afzal, Deborah S Katz, Jam Marcos Hernandez, and Claire Le Goues. Crashing simulated planes is cheap: Can simulation detect robotics bugs early? In *2018 IEEE*

11th International Conference on Software Testing, Verification and Validation (ICST), pages 331–342. IEEE, 2018.

[43] James Tu, Mengye Ren, Siva Manivasagam, Ming Liang, Bin Yang, Richard Du, Frank Cheng, and Raquel Urtasun. Physically realizable adversarial examples for lidar object detection. *CoRR*, abs/2004.00543, 2020.

[44] Patrik Vacek, Otakar Jašek, Karel Zimmermann, and Tomáš Svoboda. Learning to predict lidar intensities. *IEEE Transactions on Intelligent Transportation Systems*, 23(4):3556–3564, 2022.

[45] Jessica Van Brummelen, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation research part C: emerging technologies*, 89:384–406, 2018.

[46] Jingkang Wang, Ava Pun, James Tu, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles, 2021.

[47] Trey Woodlief, Sebastian Elbaum, and Kevin Sullivan. Semantic image fuzzing of ai perception systems. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 1958–1969, 2022.

[48] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Péter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation. In *ECCV*, 2020.

[49] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[50] Yingzhi Yang, Zhang Yan, and Brenda Goh. Baidu ceo says ev arm's autonomous driving tech will be ahead of tesla. *Reuters*, Aug 2022.

[51] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CoRR*, abs/2006.11275, 2020.

[52] Xiangyu Yue, Bichen Wu, Sanjit A. Seshia, Kurt Keutzer, and Alberto L. Sangiovanni-Vincentelli. A lidar point cloud generator: from a virtual world to autonomous driving, 2018.

[53] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[54] Zhi Quan Zhou and Liqun Sun. Metamorphic testing of driverless cars. *Commun. ACM*, 62(3):61–67, feb 2019.

[55] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. *CoRR*, abs/2011.10033, 2020.

[56] Yi Zhu, Chenglin Miao, Foad Hajiaghajani, Mengdi Huai, Lu Su, and Chunming Qiao. Adversarial attacks against lidar semantic segmentation in autonomous driving. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 329–342, New York, NY, USA, 2021. Association for Computing Machinery.