S^{3}C: Spatial Semantic Scene Coverage for Autonomous Vehicles

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

Felipe Toledo
University of Virginia
Charlottesville, VA, USA
ft8bn@virginia.edu

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

Matthew B Dwyer
University of Virginia
Charlottesville, VA, USA
matthewbdwyer@virginia.edu

ABSTRACT

Autonomous vehicles (AVs) must be able to operate in a wide range of scenarios including those in the *long tail* distribution that include rare but safety-critical events. The collection of sensor input and expected output datasets from such scenarios is crucial for the development and testing of such systems. Yet, approaches to quantify the extent to which a dataset covers test specifications that capture critical scenarios remain limited in their ability to discriminate between inputs that lead to distinct behaviors, and to render interpretations that are relevant to AV domain experts. To address this challenge, we introduce S^{3}C, a framework that abstracts sensor inputs to coverage domains that account for the *spatial semantics* of a scene. The approach leverages scene graphs to produce a sensor-independent abstraction of the AV environment that is interpretable and discriminating. We provide an implementation of the approach and a study for camera-based autonomous vehicles operating in simulation. The findings show that S^{3}C outperforms existing techniques in discriminating among classes of inputs that cause failures, and offers spatial interpretations that can explain to what extent a dataset covers a test specification. Further exploration of S^{3}C with open datasets complements the study findings, revealing the potential and shortcomings of deploying the approach in the wild.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Computer systems organization** → *Robotic autonomy*; • **Computing methodologies** → **Perception**; **Vision for robotics**.

KEYWORDS

coverage, scene graph, autonomous vehicles, perception

ACM Reference Format:
Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B Dwyer. 2024. S^{3}C: Spatial Semantic Scene Coverage for Autonomous Vehicles. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Autonomous vehicles (AVs) promise to increase safety through super-human perception and reaction ability, reduce emissions through less crowding, increase independence for those unable to drive, and other quality of life improvements [42]. Yet, automating the full scope of the generalized driving task has remained an ongoing challenge, limiting current systems to only partial autonomy under specific environments [1]. To reach viability for the generalized driving task, we must design, build, and test AV systems under the full range of scenarios they may encounter under their operational design domain (ODD) [42], including and especially the long tail of rare but safety-critical events.

The development of AV systems that can operate in such scenarios is predicated in part on the collection of rich datasets consisting of sensor inputs and expected outputs that can support training of learned components and system testing. The open question that we tackle that constitutes the developer's use case for this work is: **to what extent does a driving dataset cover the long tail distribution of relevant scenarios?**

Developing a coverage measure that can render such a judgement requires the definition of a *coverage domain*, which identifies a set of relevant elements to which a test input may be mapped. Useful coverage domains are: 1) **interpretable**, containing elements that allow engineers to understand the semantics of an input and determine what additional inputs are lacking, 2) **discriminating**, mapping complex inputs into equivalence classes when they cause similar system behavior or in different classes otherwise, and 3) **automated**, allowing them to scale to systems and inputs beyond the reach of manual methods. Coverage domains that satisfy these attributes exist for traditional software, ranging from the coverage of models to the coverage of code constructs[1].

Developing such coverage domains for the complex and high-dimensional sensor inputs that make up AV datasets is difficult. Simple domains, such as the number of inputs or the "miles travelled" acquiring inputs [4, 23, 27], while easily automated, are not discriminating or interpretable as the abstraction fails to reveal information about what occurred during those miles, or how much remains unseen. Code-coverage domains, while automated, are limited in their interpretability relative to the semantics of the complex sensor inputs. Alternatively, one could follow a requirement-based testing approach, constructing a dataset that by design provides domain coverage. For example, datasets collected through the exploration of hand-chosen scenarios including those developed by NHTSA [42], e.g. a lane change maneuver, lead to coverage domains that are interpretable and discriminating by design. Although useful for identifying and testing key scenarios, such manual efforts offer very limited coverage of the long tail of potential scenarios, saturating too quickly to be informative about whole-system safety.

Still, NHTSA [42] and EUCAR [37] offer valuable insights about what type of coverage domain would be most relevant for AVs. They observe that "It will be difficult to achieve significant coverage of the

[1]We discuss requirements, models, code, and neural coverage in § 2.

ICSE 2024, April 2024, Lisbon, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

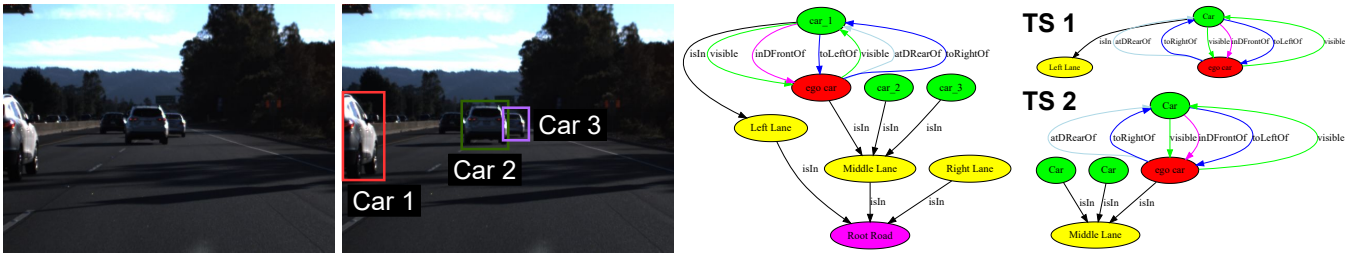

Figure 1: An input (left), its bounding boxes (middle-left), a scene graph (middle-right) that captures an abstraction of the scene that is sufficient to evaluate the preconditions of test specifications TS1 and TS2 on their sliced subgraphs (right).

variety and combination of conceivable test conditions, particularly related to ODD and OEDR." In particular, we note that *object event detection and response* (OEDR), which is a characteristic of level 3 AVs, requires the vehicle to perceive the spatial distribution of entities from the perspective of the *ego* vehicle (the one perceiving the environment). Evaluating such capabilities over a broad range of conditions is essential for high-confidence AV deployment.

Given an AV sensor input dataset, our goal is then to judge the extent that a dataset covers a coverage domain with spatial semantics as per the ODD and the OEDR specifications. Consider a narrow testing specification, *TS1*, which states that **when a car or truck is near and to the left of the ego vehicle**, then the ego vehicle should not steer left by more than δ. An appropriate coverage domain for the bolded precondition would distinguish inputs that have (1) car on the left that is near the ego vehicle, and (2) truck on the left that is near the ego vehicle. The input image in Fig. 1 provides coverage for 50% of the precondition as a truck is not present to the left of the vehicle in the image. Now consider a test specification, *TS2*, which states **when a car is very near or near in the front left, very near or near in the front center, or very near or near in the front right**, then the ego vehicle must steer away from the car. The coverage domain for the bolded precondition would distinguish inputs that have a car that is (1) very near front center, (2) near front center, (3) very near front left, etc. The image in Fig. 1 covers only 17% of the domain. Defining a coverage domain that captures a scene's spatial semantics permits the conditions that give rise to the satisfaction of a test specification precondition to be thoroughly assessed.

To quantify the coverage of such test specifications we propose the use of scene graphs (SGs) [6, 22] as the basis for a sensor-independent coverage domain for AVs that enjoys all three benefits outlined above. As illustrated in Fig. 1, an AV camera renders images from which relevant vehicles and lanes are detected, serving as nodes in the SG, and then the identified spatial relationships between those entities define the graph edges. By encoding the semantic spatial relationships between entities in a scene, SGs are **interpretable** for a broad range of system requirements such as TS1 and TS2. The definition of an SG can be customized, in terms of both its vertex and edge sets, allowing the level of **discrimination** to be tailored to the system so that inputs are grouped into equivalence classes that render similar behavior. Finally, recent advances in machine learning allow high-quality SGs to be produced **automatically** from sensor data. This intermediate representation has been recognized in prior AV work for its ability to synthesize relevant information for risk assessment [30, 52] and image synthesis [40], but has not been explored for test coverage.

The key contributions of this paper are: (1) Identification of scene graphs as an appropriate basis for an AV dataset coverage domain; (2) Definition of the S^{3}C framework which abstracts scene graphs to define interpretable and discriminating coverage subdomains for test specifications; (3) A study applying S^{3}C in simulation and for real AV datasets to explore its abilities as a test-adequacy computation technique; and (4) A publicly available open source implementation[2] of S^{3}C.

2 RELATED WORK AND BACKGROUND

This section reviews coverage families and introduces scene graphs, the basis for the proposed coverage domain.

2.1 Coverage

Coverage quantifies the extent to which a test suite exercises a system. We now discuss the kinds of coverage available for the AV domain focusing on discrimination, intepretability, and automation.

White-box structural code coverage metrics have been widely studied and adopted for their ease of use and basic interpretability for developers [20, 49, 53]. A plethora of tools to *automatically* collect code coverage metrics exist and developers readily *interpret* them as they correspond to their implementation. Research has shown how sufficiently precise structural coverage metrics, e.g. statement coverage rather than file coverage, can *discriminate* system behavior for a range of development tasks, such as fault localization [47]. However, these criteria do not provide this same utility when applied to AVs. Many AV system components have a linear control flow, e.g. parameterized control-loops or machine-learned components for perception, limiting the ability of structural coverage to discriminate relevant behaviors [18]. Prior work generalized white-box structural coverage to neural networks [34, 41]. Such coverage can be computed *automatically*, and further exploration showed its *discrimination* for the AV domain [43], but it is not *interpretable* due to the opaqueness of neural networks [26].

Black-box coverage includes a range of approaches from those focused on the domain derived directly from the requirements [2, 31, 36] to those on domains defined by system and environmental models [10, 45]. They are *interpretable* and *discriminating* by construction, but typically involve manual crafting of models and formalization of requirements to derive the inputs. Recent work on black-box coverage of neural networks relies on either (a) manually developed models of the input space [54] or (b) machine-learned

[2]https://anonymous.4open.science/r/s3c

models of the input space [11]. Manually developed models can be tailored to a system so that the elements of the coverage domain are *interpretable* and can *discriminate* between relevant system behaviors, yet the manual effort involved typically limits *automation*. For target domains for which automated mechanisms exist, as we propose for AVs given the proper abstractions, this obstacle can be overcome. Machine-learned coverage domains can also be tailored to control the level of *discrimination* and they enjoy the advantage that they are *automated*, which allows them to scale to systems beyond the reach of manual methods. Unfortunately, like most ML approaches they lack *interpretability* [26].

Several coverage metrics have been proposed to target the AV domain. The number of miles driven has been explored due to its ease in *automation*, but is not *discriminating* or *interpretable* [4, 23, 27]. Majzik et al. defined scenario coverage [28], where a scenario describes a sequence of scenes, and a scene describes a snapshot of the environment. As a form of requirements testing, such a process guarantees *interpretable* coverage of safety-critical inputs that can *discriminate* system behavior, but the substantial manual effort in designing pertinent scenarios prevents *automation* [37, 42]. Hu et al. compute trajectory coverage by examining utilization of road regions within a scenario [19]. It has limited *discrimination* power as it makes over-simplifying assumptions about the road structure, and it ignores entities. Closest to meeting our requirements is the recent work by Hildebrandt et al.'s PhysCov [18] that combines sensor data and AV kinematics to define coverage over the environment and internal system state. While this gray-box approach is *automatic* and *discriminatory* (we compare against PhysCov as a baseline for discrimination in § 4.2), describing coverage based on LiDAR-sensor values provides limited *interpretability* through just spatial information and lacking support for entity and relation semantics. No prior approach provides an *interpretable*, *automated*, and *discriminating* coverage metric for the AV domain at the input level.

2.2 Scene graphs

A long studied task in computer vision is that of object detection. Early systems could place bounding boxes around instances of a few different object classes, e.g., cars, people, buses. Modern deep learning approaches, like Detectron2 [48], are pre-trained to identify thousands of object classes and perform panoptic segmentation which identifies pixel-masks for each instance of the class.

Most scene understanding and reasoning tasks require information about the relationships between objects, e.g., one car next to another. SGs represent objects and relationships in the form of a directed graph [22] where vertices define the set of entities in a sensor input, e.g. an image or LiDAR point cloud, and edges define a set of spatial relations over the entities. SGs may also be enriched with vertex or label attributes that capture additional semantic information, e.g. car model. SGs define an abstraction of the sensor data and can be defined to capture just the information needed in a scene-based specification.

There are a growing number of scene graph generation (SGG) techniques [6]. Most SGGs consist of an object detection stage followed by a pairwise relation recognition stage. Within the context of AVs, specialized SGGs have been proposed that take advantage of domain semantics, e.g., static versus dynamic objects, road types, vehicle size, to produce more meaningful SGs [25, 29, 35] and are becoming more configurable allowing definition of object categories, e.g., vehicle types, enhanced attribute capture, e.g., direction of vehicle movement, or refinement of relationships, e.g., what distances between vehicles is considered 'near'. Even more sophisticated SGGs can track objects across video frames [51] and generate relations from natural language captions for images [50], though not specialized to the AV domain.

Among SGGs, we highlight RoadScene2Vec, an open-source toolset for generating, encoding, and utilizing road SGs specialized for the AV domain [29, 52]. Of particular interest to S^{3}C is RoadScene2Vec's ability to generate SGs from images collected in the CARLA simulator or other image datasets. The generator offers a rich set of configuration parameters that facilitate the experimentation of different pixel to SG abstractions that we leverage in our approach. For example, the configuration file includes parameters that define what entities to include (the default list are those included in the CARLA simulator such as car, pedestrian, etc.), what directional relations (pairs of object types for which relationships are generated such as ego to car, or car to pedestrian) can be instantiated and what are their thresholds (e.g., direct front ($inDFrontOf$) is [-45°, 45°], side front ($inSFrontOf$) is [-90°,-45°]∪[45°,90°]), and what proximity relations (e.g., near collision, super near, very near, near, and visible) must be instantiated also with their thresholds (e.g., [0, 4), [4, 7), [7, 10), [10, 16), and [16, 25) meters).

3 APPROACH

We formalize coverage for the AV domain, describe the architecture to compute S^{3}C, and conclude with a discussion of its applicability.

3.1 AV Coverage Domains

Given a set of sensor inputs, T, selected to represent a much larger set, D, of inputs that could be encountered in deployment, an abstraction, $f : D \mapsto A$, retains features of input data as elements of an abstract domain, A. Abstractions allow quantification of the extent to which T is representative of D relative to those features:

$$cov_f(T, D) = \frac{|\{f(x) : x \in T\}|}{|A|}$$

$cov_f(T, D)$ forms an adequacy criterion for T and helps to identify limitations in T, i.e., uncovered elements of A that developers would target with data augmentation or additional test inputs.

There are two significant challenges to make $cov_f(T, D)$ for the AV domain interpretable, properly discriminating, and automated.

The first challenge is the development of abstraction f. For AVs, we contend that abstractions must lift raw sensor inputs (e.g., camera images, LiDAR point clouds) to semantic spatial distributions relative to the ego vehicle since those ultimately inform the vehicle's behaviors in the physical world. As such, we propose abstractions that are refinements of SGs defined over entities, E, in the AV domain (e.g., cars, pedestrians) and their relationships, R (e.g., left, in front). SG abstractions, $sg(x)$, can be computed automatically and provide interpretability, but are just the first step in the input abstraction process which allows subsequent abstractions, α, to be composed, $f = \alpha \circ sg$, that discriminate relevant system behavior.

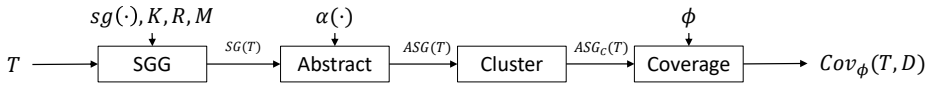

Figure 2: S^{3}C components, parameters, and ordering.

Second, A aims to define the feasible abstract domain, $\{f(x) : x \in D\}$, but this may be challenging to calculate when a precise definition of D is not available. In traditional notions of structural code coverage it is common to overapproximate the coverage domain, e.g., assuming that all pairs of definitions and uses are feasible. We adopt a similar approach by using the structure of A to compute an overapproximation, $\hat{A}$, which is then used to compute coverage. For example, an identity abstraction, $id(x) = x$, on 100 by 100 pixel images with 256 values per RGB channel could be overapproximated by the full-space of pixel combinations, $|\hat{A}_{id}| = 256^{(100\times100\times3)} \approx 10^{72247}$. This is a severe overapproximation since any realistic definition of D will comprise an infinitesimal portion of $\hat{A}_{id}$. A coarser abstraction that counts the number of cars in an input, $nc(x)$, would yield an abstract domain that could be overapproximated relatively accurately with reasonable assumptions about the deployment context, e.g., $\hat{A}_{nc} = [0, 100]$. These extreme abstractions illustrate the concept, but a more customizable framework for abstracting inputs is needed.

3.2 Architecture

The S^{3}C architecture consists of sequence of 4 components shown in Fig. 2 whose functionality is described next[3].

3.2.1 Scene Graph Generation. A scene graph generator (SGG) maps a set of sensor inputs, T, to a graph representation, $sg : T \mapsto G$. SGGs can be parameterized to define how they interpret sensor data. For example, the set of entity kinds, K, represented by graph vertices, the relationships among entities, R, that define edges in the graph, and attribute values, M, associated with edges and vertices. More formally, a scene graph,

$$G = (V, E : V \mapsto V, ego \in V,$$
$$kind : V \mapsto K, rel : E \mapsto R, att : V \cup E \mapsto M)$$

is a directed graph with a distinguished *ego* vertex and functions to access the entity *kind* of a vertex, the *rel*ation encoded by an edge, and *att*ribute values of vertices and edges. A map M is used to associate attribute values with each type of attribute.

Example. RoadScene2Vec [29] uses a forward facing camera image to generate a scene graph. A configuration file enables the user to tailor the SGs to their specific ODD and AV characteristics by specifying the entity and relationship kinds. For example, if the ODD is a rural environment, the entity list may include tractors [4].

3.2.2 Abstraction. Abstractions transform SGs to retain salient information for analysis. An SG abstraction, $\alpha : G \mapsto G$, allows the discriminating power to be refined. Since $\alpha \circ sg$ defines an abstraction of the input sensor data, α defines a coverage domain, $\{\alpha(sg(x)) : x \in D\}$, for sensor datasets. Moreover, as we shall see in the study, different abstractions may be defined and composed to provide alternate coverage measures for analyzing a given dataset.

Separating generation from abstraction offers several advantages. First, off-the-shelf SGGs can be reused without modification. Second, different SGGs, perhaps using different sensors, could be used to generate more accurate SGs. Third, abstractions can be defined and reused across systems. Finally, an appropriate composition of abstractions can be selected to suit developer needs.

Example. The SG vertices generated by RoadScene2Vec include an identifier attribute associated with each entity to differentiate among multiple instances of an entity class. Such identifiers are superfluous to the spatial distribution of entities a coverage domain is meant to capture and can be abstracted from the graph. For example, the left of Fig. 1 shows 'car_2' and 'car_3' in the middle lane—the spatial relationships would be identical if they were relabelled 'car' as shown on the right. An *id*entifier suppressing abstraction might be defined as $\alpha_{no-id}(g) = g'$, where $g = g'$ except that $\forall v \in g.V : v.kind = car \implies g'.v.att.M[id] = \perp$. The space of possible abstractions includes ones that can transform the structure of the graph. For example, the vertices in the abstraction could be restricted to those that lie on paths of length k from the ego vehicle as follows: $\alpha_{khood}(g) = g'$ where $g = g'$ except that $g'.V = \{v : v \in g.V \wedge \exists e_1, \ldots, e_k \in E : v \in e_1(ego) \vee \ldots \vee v \in e_k(\ldots(e_1(ego)))\}$ and $g'.E$ is restricted to edges in g incident on $g'.V$.

3.2.3 Abstraction Clustering. Given dataset, T, and abstraction, α, the collection of abstracted scene graphs, $ASG(T) = \{\alpha(sg(x)) : x \in T\}$, is the set of scene graphs computed over T. The multiset of scene graphs represents equivalence classes of isomorphic graphs:

$$ASG_C(T) = \{C \mid C \subseteq ASG(T) \wedge$$
$$\forall c_i, c_j \in C : c_i \cong c_j \wedge$$
$$\forall c_k \in C, \forall c_\ell \in ASG(T) - C : c_k \ncong c_\ell\}$$

Each $C \in ASG_C(T)$ is the maximal set such that all graphs in C are isomorphic to each other and not isomorphic to any other graph in $ASG(T) - C$. $|C|$ gives the number of times the abstract scene graph occurs in the dataset. If no pair of ASGs are isomorphic, then $|ASG(T)| = |ASG_C(T)|$, and each ASG is its own class. However, in most cases, clustering will result in a reduction in the number of graphs to consider such that $|ASG_C(T)| < |ASG(T)|$.

In general, performing the clustering is computationally expensive because of its reliance on graph isomorphism; there is currently no known polynomial algorithm for determining if two graphs are isomorphic [3, 14]. The worst case scenario requires testing isomorphism across all pairs of ASGs and is $O(|T|^2)$. However, in practice we can exploit the data's long tail distribution to drastically reduce the number of computations. We first use a hash table to group the ASGs using a hash based on easy-to-compute graph statistics, such as number of entities (nodes) and relations (edges) or number of entities by kind which can be done in $O(|T|)$ for the average case.

[3]Pseudocode for each component is available in the online repository.
[4]Discussion on SGG guidelines for practical use available in the online repository.

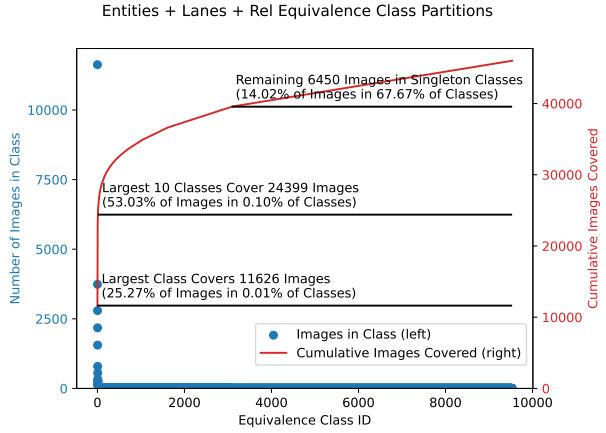

Figure 3: Distribution of images across scene graph equivalence classes for the *ELR* abstraction studied in § 4.2.

Then, the final equivalence classes are computing by pairwise isomorphism testing across these smaller groups, reducing the number of computations and allowing for parallelization. With additional optimizations such as special handling for the empty abstraction, computing the clustering can be performed in practical time scales.

Example. Fig. 3 visualizes the count and size of the equivalence classes of an abstraction, *ELR*, further discussed in § 4, that retains information about entities, their relationships, and their locations within the road lanes to analyze an image dataset, T. Here, $|ASG(T)|$=46,006 and $|ASG_C(T)|$=9,532—a reduction of almost 80%. This also shows the imbalance in the equivalence classes as the largest class, the class of an empty single-lane road, contains 25% of the dataset, in other words, a quarter of the dataset presents no diversity in terms of spatial distribution. The largest 10 classes contain more than half of the dataset, and the largest 6, containing 22,684 images (49.3%) represent different lane configurations with no other entities besides the ego vehicle. Additionally, we can see the effect of the long tail of the distribution as 6,450 dataset images are in singleton classes, meaning more than $^2/_3$ of the classes contain only a single image. Computing this $ASG_C(T)$ took 1047 seconds[5]. § 5 discusses time performance for additional datasets.

3.2.4 Coverage Computation. This component refines the notion of interpretability introduced earlier by considering the matching between the abstraction, $\alpha \circ sg$, with the preconditions of test specifications, $\phi_\alpha : D \mapsto Boolean$ (we refer to ϕ_α as ϕ when α is clear by its context). Let A be the codomain of $\alpha \circ sg$—the set of possible abstracted scene graphs. We say that A is *interpretable* relative to ϕ when it reflects the semantic distinctions made by ϕ in the input domain:

$$\phi_\alpha(a) \iff \forall x \in D : a = \alpha(sg(x)) \implies \phi(x)$$

For an interpretable abstraction, coverage of a test specification precondition, ϕ, is the portion of the space defined by ϕ that is exercised by $\alpha(T)$. To determine coverage, ϕ is projected onto A to compute a subspace, $A_\phi \subseteq A$, that includes all possible ASGs that satisfy ϕ. Satisfaction of ϕ depends on the subset of the relations, entities, and attributes in A that are referenced in ϕ. Many SGs may exist that satisfy ϕ, but vary only in features of A independent of ϕ.

[5]30 threads. 32 logical core Xeon 4216, 128GB of RAM

We apply a form of model-based *slicing* [24] to reduce the coverage domain from A_ϕ to include a single representative of each subset of A_ϕ that varies only in features on which ϕ is dependent. For each clause in the disjunctive normal form encoding of ϕ, we slice based on the clause, and eliminate any portion of the slice that is not reachable from the ego vehicle in the sliced scene graph $\psi(a)$:

$$\psi(a) = \{a' : a' = reach(slice(a, \phi'_\alpha), ego) \wedge \phi'_\alpha \in clause(\phi_\alpha)\}$$

The coverage domain is defined $S = \bigcup_{x \in D} \psi(\alpha(sg(x))$; for brevity, we say $S = \psi(D)$. This allows coverage to be computed for a test specification as:

$$cov_\phi(T, D) = \frac{|\psi(T)|}{|\psi(D)|} = \frac{|\psi(T)|}{|S|}$$

To establish a finite coverage domain we bound the number of vertices in an SG—in practice SGGs can only produce bounded SGs. Note that S may overapproximate the coverage domain, but it is a tight approximation when ϕ is purely conjunctive or consists of disjunctions of independent clauses. This is because the former reduces to a singleton coverage domain under slicing, i.e., **there is a car in front of the ego vehicle**, and the latter can be computed as the product of the valuations of each independent clause.

Example. Consider the test specification precondition, ϕ: **the closest lane to the left of ego contains a car at distance: d_1, …, or d_n**; i.e. for an input, $i \in D, \phi(i) \iff i$ contains a car in the closest lane to the left of the ego at distance $d_1, \ldots,$ or d_n. Like all test specifications, it is stated relative to the ego vehicle. The precondition establishes bounds on the number of lanes and the number of cars in subgraphs which satisfy it. Specifically, there is 1 lane that is the closest lane to the left of the ego, and it may contain up to n cars, one each at distance d_i from the ego vehicle. There are 2^n possible subgraphs that can be constructed from the direction, containment, and distance relations mentioned in this specification. One of those subgraphs has no cars, which falsifies the precondition, leaving $|\psi(D)| = 2^n - 1$ satisfying subgraph models to cover. In this example, because different cars can appear at different distances independently, the approximation of the coverage domain is tight. We will see specifications in § 4.3.1 for which this approach suffices. We leave to future work the problem of computing tight approximations of the coverage domain for test specifications with dependencies among precondition clauses.

3.3 Applicability of Coverage Values

We identify 3 potential failure modes that can limit S^{3}C applicability.

Abstraction-Sensor Mismatch. The abstraction design must align the coverage domain with the physical capabilities of the system's sensors. That is, the system's sensors must be sufficiently sensitive to capture the dimensions of the abstraction. For example, an abstraction domain that includes vertex attributes based on the color of an entity produces an immeasurable distinction if the system is using a LiDAR sensor which cannot sense color. Another example is a camera with a shallow depth of field that blurs far-away entities. This could obscure entities and distort distance estimates, limiting the entities and relationships in the abstraction. These limitations can lead to portions of the abstraction domain being impossible to cover, causing coverage to appear artificially low. This lack of sensitivity can also affect the interpretation of the abstract domain. Consider an abstraction intended to count the number of cars. Given

a shallow depth camera, the abstraction more accurately defines the number of cars within a shallow distance not occluded from the camera, which alters the interpretation of the abstraction.

Leaky Abstractions. Designing an abstraction domain requires careful consideration to ensure that the retained information is sufficiently specific to differentiate between inputs leading to distinct system behaviors. For example, the abstraction *nc* that counts the number of cars in the scene does not provide sufficient specificity to appropriately differentiate system behavior as seeing a car parked off the road and parked in the ego vehicle's lane both have $nc(x) = 1$, but should elicit completely different behaviors. This can lead to artificially high coverage as the coverage domain is not capturing the relevant details of the problem domain. We explore the level of abstraction specificity required for utility in § 4.2.

Perception Failures. The process of automatically mapping sensor inputs to the abstraction domain may introduce inaccuracies where the input is not mapped to the intended SG. An incorrect mapping can occur due to failures in sensing or perception, and can cause it to incorrectly identify entities, not perceive entities, or perceive entities that do not exist—§ 5 delves into this analysis. These failures can both increase and decrease coverage as two inputs that should be mapped to the same SG could be separated, or vice versa. Although problematic, these failures can be mitigated, e.g. Detectron2 provides a confidence threshold that can be tuned to reduce the number of inaccuracies, and RoadScene2Vec provides parameters to redefine the relationships. Further, combining multiple different sensors or perception techniques, or sequences of sensor values, can reduce perception failures [13]. The rapid evolution of techniques for SG generation, particularly when specialized for a domain like AVs, the suggested mitigation strategies, and S^{3}C parameterization offer a rich space of tradeoffs to overcome these challenges.

4 STUDY

This study explores the following research questions:

- RQ#1: How effective is S^{3}C at discriminating inputs that lead to different behaviors? More precisely, we assess the discriminating capabilities at different levels of abstraction in terms of equivalence classes of scene graphs covered during training versus those failing during testing.
- RQ#2: How effective is S^{3}C to assist in the interpretation of the differences between datasets and the coverage achieved for various dataset subsets with regard to test specifications?

4.1 Setup

To answer the research questions we designed an experiment. The units of analysis are datasets consisting of sensor inputs captured from an AV operating in simulation. The treatments are the approaches to abstract the inputs from a dataset into classes that correspond to a coverage domain. The dependent variables correspond to the ability of the treatments to discriminate and interpret those sensor inputs. The hypothesis is that S^{3}C will outperform alternative approaches, and that the best instance of S^{3}C's approach will be able to assist in the discrimination of failing inputs not seen in training, in the analysis of those failures, and in the interpretation of the extent to which test specifications are covered by a dataset. We provide details on these aspects next.

4.1.1 Dataset. A dataset of sensor inputs was collected by launching a pre-programmed AV in a simulated environment. We use the CARLA simulator [12], a state-of-the-art driving simulator that is widely used in AV development and testing. CARLA provides multiple preconstructed environments designed to mimic various driving environments. We utilized 4 environments, called "towns" in CARLA, to cover a mix of suburban, urban, and highway driving and roads with 2, 4, and 8 lanes. To vary the spatial structures observed in the simulations, we run simulations under two conditions: zero vehicles and max vehicles. In the zero vehicle scheme, the ego vehicle is the only dynamic vehicle on the roadway. In the max vehicle scheme, we utilize CARLA's autopilot feature to place and guide the maximum number of vehicles the environment can support—ranging from 101 to 372 vehicles for the towns chosen. The vehicles are generated to be 80% car and 20% truck, each randomly sampled from the 25 car and 4 truck models available. Using the CARLA Python API, we recorded at 5Hz: an RGB sensor input from a single camera mounted on top of the ego vehicle, the steering commands supplied by the ego vehicle's autopilot, the orientation and position of all vehicles, and road structure details, e.g. lanes, curvature, etc. For each of the 2 traffic schemes and 4 maps, we executed 15 tests 5 minutes in duration with different random seeds.

4.1.2 Steering AV. Since the study aims to assess the ability of S^{3}C to discriminate passing from failing behaviors with respect to a coverage domain, we built an AV steering model that consumes images to steer the vehicle; this allows us to record which inputs have been covered, i.e. seen by the model during training. We trained an AV using a PyTorch [33] model based on ResNet34 [17] pre-trained on ImageNet [9] with the last layer modified for predicting a steering angle to learn to mirror the behavior of the CARLA autopilot. This allows us to use the CARLA autopilot as both the source of training data and the oracle of test correctness for the learned system. As this task does not involve traffic decisions, we remove all observations that involve approaching or transiting intersections, and prevent the ego vehicle from changing lanes. Further, $\approx$ 5% of data points were removed due to simulation failures in CARLA (sensor timing mismatch, rendering error, etc.), resulting in a dataset of 46,006 observations partitioned into 80% (36,809) training and 20% (9,197) test inputs[6]. When evaluating model performance, we designate a failure as a steering prediction more than 5° from the ground-truth. An error threshold of 5° is sensible because at 25mph, turning 5° off normal results in the vehicle fully crossing into an opposing lane in less than 1 second. This leads to 17 failures of the 36,809 training inputs (0.05%) and 362 failures of the 9,197 test inputs (3.94%).

4.1.3 Treatments. As discussed in § 3.3, SGGs are rapidly improving and are being tailored to the AV domain. Still, they are sophisticated systems with their own failure modes and complex configuration spaces. Working in simulation allows us to control for the quality of SGs as a coverage domain[7]. To explore the effect of SG expressiveness, we implemented five SGG-based abstractions including various levels of semantic information. Each abstraction uses ground-truth data from CARLA's internal state and includes all

[6] Additional training details available in the online repository

[7] § 5 provides a complementary exploration using existing SGGs on real-world datasets.

Table 1: Scene graph abstractions studied. $|Img| = 46{,}006$

Abstraction	Short Description	$\lvert ASG_C \rvert$	$\lvert Img \rvert / \lvert ASG_C \rvert$
Entities (E)	Semantic segmentation	305	150.8
Entities + Lanes (EL)	E and ground-truth lane occupation for each entity	2,932	15.7
Entities + Relations (ER)	E with RoadScene2Vec's default inter-entity relationships configuration	8,821	5.2
Entities + Lanes + Relations (ELR)	E with both lane and relationship information from EL and ER.	9,532	4.8
Entities + Road Structure (ERS)	EL, except lanes are modeled as multiple detailed road segments	22,987	2.0

entities within a 50m by 50m square, horizontally centered on the ego vehicle and offset to include the 45m ahead of the ego vehicle.

To evaluate the effect of the SGG-based abstraction considered, we explore five abstractions of varying specificity as outlined in Table 1. The weakest abstraction uses only the list of entities in the scene without any spatial information, providing a baseline for our experiments emulating what entity detection tools provide. We additionally explore three abstractions inspired by the abstraction used in RoadScene2Vec [29] that add information about lanes and inter-entity relationships. We utilize the default inter-entity relationship configuration of RoadScene2Vec (introduced in § 2.2) to add spatial relationships to the graph—full details of the configuration space are available in the online appendix. In its original abstraction, RoadScene2Vec assumes that a road can always be characterized into three lanes: 'Left', 'Middle', and 'Right' with the ego vehicle always in the 'Middle' lane. We remove this assumption and strengthen this abstraction, creating separate nodes for each logical lane in the graph; the lane containing the ego vehicle is labeled the 'Ego Lane' with lanes to the right or left traveling with the ego lane labeled 'Right Lane 1', 'Right Lane 2', etc. Additionally, any other lanes present are labeled 'Opposing Lane x' with x between 0 and the number of other lanes present. In our experiments, the lane structure is recovered using the ground-truth simulation data, though it could come from high-definition maps in use in modern AVs [46] or from onboard sensing [32]. Finally, we explore an abstraction that models lanes as a sequence of nodes that preserve the road structure in 2.5m increments, e.g. a 10m section of two-lane road becomes a directed graph on 10 nodes, 4 per lane, where each node represents a 2.5m length of a lane and has edges to the lane segment that traffic will flow to and the segment corresponding to a lane change; each segment also has a curvature attribute. We add an edge between each entity and the lane segment it occupies; we do not include inter-entity relationships as many relationships can be approximated from the graph-path between entities through the road; future work should explore adding additional relationships. This refines the granularity of the abstraction and thus increases the number of equivalence classes generated.

4.2 RQ #1: On Discrimination

In this section we utilize an instance of S^{3}C that includes multiple abstractions to assess its discriminating capabilities and tradeoffs.

Metrics. For S^{3}C to provide utility it must *discriminate* among complex inputs that cause distinct system behaviors. We partitioned the dataset so each input is either used for training or testing, and based on the AV model studied each input results in either a pass or fail. To analyze the discriminating ability of S^{3}C as a coverage metric, we measure how well it discriminates between passing training inputs and failing test inputs. We compute the **P**ercentage of testing inputs that cause **N**ovel **F**ailures that reside in an ASG

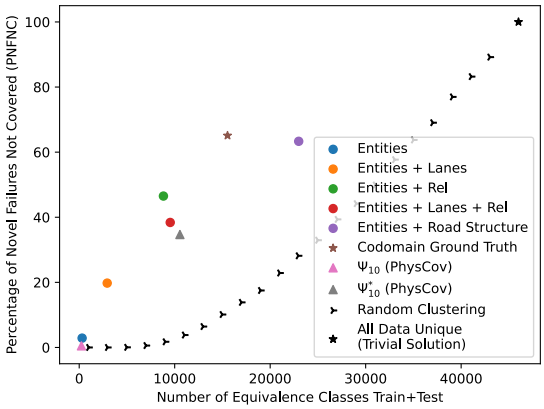

Figure 4: Percentage of novel test failures in an equivalence class not covered in training (PNFNC) versus count of equivalence classes under different abstractions. High percentage and low total classes is better.

equivalence class that was **N**ot **C**overed during training under a given abstraction, which we refer to as the PNFNC metric. A novel testing failure is one that resides in an equivalence class with no training failures; given the 17 (0.05%) training failures, there may be test and training failures that are equivalent and thus the metric should not penalize grouping these failures together. A low PNFNC means the abstraction provides limited discrimination to isolate the failure-inducing inputs from those covered during training, while a high PNFNC indicates high discrimination. However, it is important to note that an abstraction that generates a higher number of equivalence classes is likely to better discriminate between inputs. In the extreme case, a scheme that considers all inputs distinct trivially results in 100% PNFNC, at the detriment of interpretability. Thus, we also examine the total number of equivalence classes needed to partition the dataset; an abstraction aims to maximize PNFNC while minimizing the total number of equivalence classes.

Baselines. Besides the five SG-based abstractions described in § 4.1.3, we instantiate 3 baselines. First, the Ψ_{10} algorithm from PhysCov [18], a LiDAR-based technique for clustering AV inputs. Since its default parameterizaton produces a relatively small number of classes, we devised a parameterization Ψ^*_{10} with increased precision that generates a comparable number of classes to the SG-based abstractions. Second, we cluster the inputs based on the ground-truth steering output. We refer to this as the 'Codomain Ground Truth' since it groups inputs based on the *expected output* and thus is not usable in practice; as this relies on ground-truth output information, it can be viewed as a test equivalency oracle from the output perspective. Third, we show the average performance

of an algorithm that, given a max number of equivalence classes, uniformly at random chooses a class for each data point[8].

Results. As seen in Fig. 4, the weakest SG-based abstraction, *E*, achieves a PNFNC of 2.9% from 305 classes, performing markedly better than the similarly-situated original PhysCov's (Ψ_{10}) performance of 0.5% from 235 classes. The four abstractions that consider spatial information, *EL*, *ER*, *ELR*, *ERS*, provide a PNFNC that outperforms the random baseline and have a substantially greater PNFNC. The *ER* abstraction yielded 215 novel failures, of which 100 were not covered for a PNFNC of 47%, while the *ELR* abstraction achieves a PNFNC of 101/263=38%. These both outperform PhysCov's Ψ^*_{10} PNFNC of 119/343=35%. We note that although the PNFNC decreases across these three techniques, the number of novel failures and those uncovered both increase, indicating that the technique is discriminating behavior less efficiently. This provides a trade-off between the number of classes, the interpretability of the classes, and the overall performance.

We observe that the most precise SG-based abstraction *ERS* achieved a PNFNC of 63% while using 22987 classes compared to the 'Codomain Ground Truth' PNFNC of 65% while using 15532 classes, demonstrating the capability of rich graph abstractions to discriminate system behavior while pointing to potential future refinements to reduce the number of classes used. Further, this increased capability to *discriminate* behavior comes at the expense of *interpretability* as the richer abstraction requires more effort to comprehend. The average size of the representative ASG of equivalence classes for *ERS* has 131 nodes and 264 edges compared to 16 nodes and 47 edges for *ELR*—this substantially smaller graph space may be much easier for developers to quickly interpret and utilize.

We observe that S^{3}C performs better than the random baseline across all abstractions, and slightly better than existing baselines, i.e. PhysCov, while adding a new dimension of interpretability. These results demonstrate the ability for S^{3}C, particularly when implemented with semantically rich graph abstractions, to *discriminate* between inputs that lead to different system behaviors.

4.3 RQ #2: On Interpretation

For this research question, we investigate the interpretability of classes generated by S^{3}C. We focus on the *ELR* abstraction as it provides a balance between PNFNC and the number of classes based on Fig. 4 while providing a sufficiently rich semantic basis for expressing the test specifications studied in Table 2. We analyze the classes from two interpretation perspectives. First, we compare the ASGs of testing failures against the ASGs of the training passes used in RQ#1, to determine what particular spatial elements contribute to test failures. Second, we explore the extent to which the datasets associated with the CARLA towns cover a set of specifications' pre-conditions defined in terms of scene spatial structure.

4.3.1 Differentiating Training Images from Failure-causing Testing Images. We aim to identify the features that differentiate the failure test set from the training pass set. To achieve this, we build and train a decision tree classifier to determine whether a given ASG belongs to the PNFNC failure test set or the training pass set. The tree nodes provide insights into the features that significantly

[8]Details and implementation of random and Ψ^*_{10} available in the online repository.

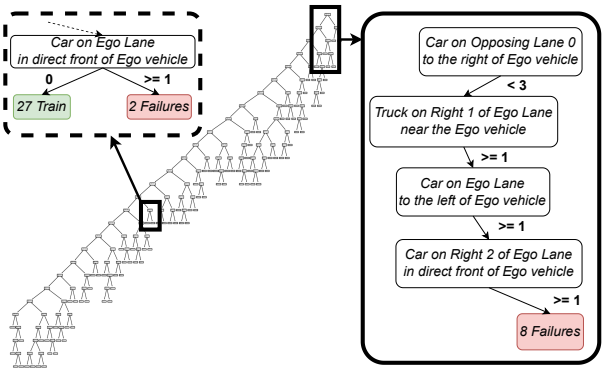

Figure 5: Decision tree to characterize failures.

Figure 6: Testing images that resulted in failures.

contribute to test failures and distinguish them from non-failing inputs. The tree is fed with a feature vector of size 484, encompassing all possible combinations of the main features extracted from the ASGs, namely, 4 entity kinds, 11 lanes, and 11 relationships. Each element in the vector represents the number of occurrences of a specific combination within that particular ASG. We trained a decision tree classifier that let us identify almost 95% (68 out of 72) failure classes with ASGs that were not seen in training.

We find that a few spatial semantics are associated with multiple testing failures studied. For example, as shown in Fig. 5 (right box), there are 4 conditions that need to be met to distinguish 8 failures from the passing inputs. More specifically, images where there are less than 3 cars on the opposing lane, at least one truck on the immediate right and near, a car on the ego lane to the left of the ego, and a car on the right-2 lane that is in direct front of the ego, were not observed during training and led to testing failures 8 times. In other words, when there are cars surrounding the ego vehicle in the on-going lanes and the opposing lane traffic is not high, the system struggled. Fig. 6 (left) shows one of the testing images (annotated to ease interpretation) corresponding to that ASG class.

Identifying the differences between an image in testing causing a failure from the training ones, however, is often more subtle. On average, test failures required 11 predicates to be differentiated from the training ones. For example, the predicate shown in Fig. 5 (left box) is the 11th, where the final classification is made. For this predicate, at least one car must be directly in front of the ego vehicle on its lane. Fig. 6 (right) shows an example of this image inducing failure, with 2 vehicles in front of the ego car.

These findings indicate that there exists not just a long-tail distribution of ASGs, but also that the difference between passing and failing may appear in just one of the hundreds of spatial features we explored. Thus, future augmentation techniques must strive to equip training datasets with such one-off scenarios.

Table 2: Testing specifications' preconditions and their coverage.

Preconditions	$\|A\|$	Coverage				
		Town01	Town02	Town04	Town10HD	Full Dataset
ϕ_1: There is a truck that is either *inDFrontOf* or *inSFrontOf* of the ego and at distance: *near_coll*, *super_near*, *very_near*, *near*, or *visible*.	242	19 (7.85%)	5 (2.07%)	3 (1.24%)	6 (2.48%)	20 (8.26%)
ϕ_2: The closest lane to the left of ego contains a car at distance: *near_coll*, *super_near*, *very_near*, *near*, or *visible*.	31	0 (0.00%)	0 (0.00%)	24 (77.42%)	23 (74.19%)	26 (83.87%)
ϕ_3: The closest lane to the left of ego is empty and there is a car or truck that is either *inDFrontOf* or *inSFrontOf* of the ego and at distance *near_coll* or *super_near*.	24	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
ϕ_4: The 11 possible lanes[9] have at least a car or a truck.	22	4 (18.18%)	4 (18.18%)	22 (100.00%)	10 (45.45%)	22 (100.00%)
ϕ_5: The 7 possible on-going lanes[9] have a car, a truck, or no vehicle.	426	3 (0.70%)	3 (0.70%)	149 (34.98%)	16 (3.76%)	163 (38.26%)

4.3.2 Coverage of test specifications. Previous findings showed the presence of distinct scene compositions among most failures, pinpointing weaknesses in the training dataset to capture diverse driving scenarios seen in deployment. Achieving coverage across all the possible scene combinations envisioned earlier can be difficult and costly. Instead, a developer may opt to strategically focus on smaller combinatorial spaces relevant to safety-critical scenes. We investigate this strategy by drafting five test specifications that include a precondition about the AV spatial semantics, and then analyze the extent to which a dataset covers those preconditions.

Findings. The specifications' preconditions, their coverage domain $|A|$, and their corresponding coverage per town and for the union of all towns is shown in Table 2. ϕ_1 gets single-digit coverage across all towns. This happens because the dataset was collected using the CARLA driver which, with its builtin conservative behavior, attempts to avoid approaching vehicles within 2m, resulting in a dataset without many near_coll and super_near relationships with other vehicles in the Ego Lane. ϕ_2 gets 0% coverage for Town01 and Town02 which is expected due to the absence of a left lane in these towns. ϕ_3 gets 0% coverage across all towns due to several factors such as the lack of left lane for Town01 and Town02, the left lane was infrequently empty and the conservative nature of the CARLA built-in driver. As is, the dataset does not provide enough images to assess how the system would perform under the safety-critical conditions mentioned in ϕ_1 and ϕ_3. On the other hand, ϕ_4 and ϕ_5 highlight the influence of the road topology. Since Town01 and Town02 only have 2 lanes, they provide a coverage of 18.18% for ϕ_4, while Town10HD with 5 lanes attains 45.45%, and Town04 with 11 lanes reaches 100% coverage. Similar findings apply to ϕ_5.

As expected, the general trend indicates that more datasets (towns) lead to greater diversity of scenarios, resulting in increased coverage. However, it is clear that not all towns contribute equally to all preconditions. Furthermore, the coverage remains low for several preconditions, with a fundamental one like having a vehicle close ahead entirely missing. The lack of coverage for critical preconditions highlights the significance of our approach, as it enables us to identify particular gaps in the datasets which can then guide the augmentation for addressing its deficiencies.

4.4 Threats to validity

The external validity of our findings are affected by our use of simulation to explore the application of S^{3}C, which may not fully generalize to real-world data. The use of simulation allowed us to instantiate S^{3}C with perfect information of the scenario including zero-noise sensing and the ability to include occluded objects in the SGs. While SGG precision will increase as perception technology advances, no real-world system will be able to fully match the performance of the simulated SGG. Further, although CARLA is widely used in the AV testing and development space, the simulation-reality gap [21] remains a limitation in generalizing results to real-world performance. While we believe the demonstrated discriminating and interpretability results of S^{3}C are encouraging, future work should investigate the ability for abstraction and automation of S^{3}C using real-world data. We take the first step in that direction in § 5, exploring S^{3}C on existing open datasets.

Further, although the AV steering model used to classify behaviors is similar and followed similar training techniques as prior work, the model was selected, configured, and trained by the authors as part of the study. AVs with input-causing failures less relevant to spatial semantics may render different results. Also, while the AV steering task has been extensively studied in prior literature [15, 34, 38, 54], it does not capture the full driving task. We focus only on binary pass-fail outcomes based on a threshold steering angle error to control the scope of the analysis; future work should explore a more refined analysis of the continuous error space. Additionally, our model-based study only validates S^{3}C's performance on single-instant inputs, e.g. a single camera image, which may not generalize to multi-frame inputs and system-level failures [16] An initial exploration into using S^{3}C with multi-frame inputs is available in the online repository.

The internal validity of our findings may be affected by the implementation of S^{3}C as it involves several complex internal and external components including the integration with CARLA, the generation of the abstractions, and the clustering of the SGs to perform the tasks outlined in the evaluation; besides carrying out extensive validation, we share the code to mitigate this threat.

In terms of construct validity, although S^{3}C exhibited interpretability as per the direct connection to test specifications, we have not validated these interpretations in the hands of domain experts to improve their datasets, refine their specifications, or investigate failures. Similarly, the findings for discrimination for input inducing failures is subject to the subtle and often noisy relationship between coverage and fault distribution. Additionally, exhibiting coverage may be valuable even for large coverage domains for which $ASG(D)$ cannot be accurately estimated.

Table 3: Equivalence Classes generated using RoadScene2Vec (left) and Entity Statistics for Open AV Datasets (right)

Dataset	Environment	Resolution	Images		$\lvert ASG_C \rvert$		$\lvert Img \rvert / \lvert ASG_C \rvert$	Average count of entity kind per image				
			#	%	#	%	Ratio	Person	Bicycle	Motorcycle	Bus	Car
CommaAi [39]	Highway	320x160	522,434	83%	27,835	57%	18.77	0.10	0.00	0.02	0.00	2.26
NuScenes [5]	Urban	1600x900	16,095	3%	8,877	18%	1.81	1.53	0.07	0.04	0.18	3.01
Sully [7]	Suburban	455x256	45,568	7%	8,363	17%	5.45	0.14	0.00	0.00	0.01	2.31
Udacity [44]	Highway	640x480	39,422	6%	6,251	13%	6.31	0.07	0.00	0.01	0.07	3.10
Cityscapes [8]	Urban	2048x1024	5,000	1%	4,902	10%	1.02	4.04	0.97	0.15	0.16	7.65
Union			628,519	100%	48,956	100%	12.84					

Table 4: Computation time for each dataset

Dataset	SGG[10] (hr)	SGG per Img (s)	Abstraction+ Clustering[11] (hr)	Abstr.+Clust. per Img (s)
CommaAi	115.5	0.80	3.92	0.03
NuScenes	9.6	2.16	0.80	0.18
Sully	11.1	0.88	0.21	0.02
Udacity	10.3	0.94	0.38	0.03
Cityscapes	7.2	5.19	0.45	0.32

5 EXPLORING S^{3}C ON REAL DATASETS

We now explore the usage of S^{3}C utilizing an off-the-shelf SGG called RoadScene2Vec that implements its own abstraction (described below) similar to *ELR* on real-world datasets from the AV literature to gauge the potential of the approach and current challenges for its application in the wild. We selected five open-source datasets from the AV training and testing literature shown in Table 3 of different sources, sizes, environments, and image resolutions.

As described in § 2, RoadScene2Vec is an SGG that uses a single RGB image to generate an SG that captures entities such as cars, trucks, and people, and spatial relationships such as left, right, near, and far along with information about which lane an entity occupies. However, we observe that RoadScene2Vec exhibits two of the failure modes described in § 3.3. First, the lane abstraction is leaky as it always assumes a three-lane structure of "Right", "Middle", and "Left", which may not match real-world data; as seen in Fig. 1, there are three lanes identified for a four lane road and car_4 is marked as being in the left lane. This limitation lessens the interpretability of the image. Second, RoadScene2Vec encounters perception failures leading to inaccurate SGs. We performed a cursory examination comparing one of the author's image perception to that of RoadScene2Vec under its default configuration on 15 images. We found that 9/15 (60%) images exactly matched, 1 had one error, and the remaining 5 (34%) contained multiple errors. Through this examination we identified that a key challenge in applying S^{3}C on real-world data is reducing the types of failures described in § 3.3 by refining the abstractions deployed and improving the perception capabilities of underlying components. We note that the reported performance was under RoadScene2Vec's default configuration, which serves as a lower bound of performance. Better parameter tuning may eliminate some perception failures, while reducing the effects of the leaky abstraction requires more fundamental changes.

Notwithstanding the potential weaknesses of RoadScene2Vec, we explore the scene graph based equivalence classes it generated for the datasets. The left portion of Table 3 shows the number of classes per dataset and their union. We note several interesting trends in this data. First, the number of equivalence classes does not increase proportional to the number of images. The CommaAi dataset, although more than 10 times larger than any other dataset, only had 3 to 4 times as many equivalence classes. This highlights the diminishing returns of collecting additional data without regard for what data is being collected—without attempting to sample from the long tail distribution the collection will cause for a coverage to quickly saturate. On the other extreme, smaller but carefully crafted datasets like NuScenes and Cityscapes have a low ratio of dataset size to ASG_C, meaning they are capturing distinct portions of the coverage domain more efficiently. Second, the average number of images per equivalence class is negatively correlated with the resolution of the images in the dataset. We hypothesize that the lower resolution limits RoadScene2Vec's ability to perceive entities in these images, leading to less rich abstractions.

The right side of Table 3 further reports on the interpretability of the abstraction in terms of the entities per image (similar analysis could be performed at the relationship level, or at the entity and relationship level). We observe that NuScenes and Cityscapes contain many more people and buses than the other datasets. This is expected since they are from urban settings unlike the others, but we also see that Cityscapes has substantially more people and bicycles than NuScenes, indicating that Cityscapes was sampled from an area with higher pedestrian traffic. This highlights the ability for S^{3}C to automatically provide interpretability in comparing the coverage across datasets independent of an AV.

Last, we discuss the efficiency of the S^{3}C implementation to process these datasets in Table 4. The SGG generation time is proportional to the number of images and their resolution, while the abstraction and clustering approach is proportional to the number of scene graphs generated by the SGG. For the largest dataset with over half a million images, CommaAi, generating the scene graphs took over 4 days. For this dataset with low resolution images, scene graph generation, abstraction, and clustering can be performed in under a second per image. For the datasets with the largest resolution, the time per image increases to up to 5.2 seconds which is only one order of magnitude slower than the time to collect the image, and will likely decrease with advances in the state of the art in perception. Overall, the findings indicate that even an unoptimized implementation of S^{3}C can scale to existing datasets.

6 CONCLUSION

We have introduced S^{3}C, the first specialized framework for the AV domain that takes a sensor input dataset and computes its coverage as per the spatial scene semantics. The framework is unique

[9] In CARLA's towns there are a maximum of 4 opposing and 4 on-going lanes. Depending the location of the ego vehicle, the on-going lanes have 7 unique encodings (i.e., if the ego lane is at the most left lane then there are 3 right lanes, if the ego lane is at the most right lane, then there are 3 left lanes).

[10] No parallelization. System with 32 logical cores, >64GB of RAM, and 4 GTX1080Ti

[11] 64 threads. System with 48 logical cores and 512GB of RAM

in how it leverages multiple levels of abstraction to automatically transform raw sensor inputs into scene graphs and then into equivalence classes that are interpretable to AV developers and that can discriminate among distinct behaviors. To assess S^{3}C, we conducted an experiment with an AV in simulation that showed that the equivalence classes generated by the approach can (1) discriminate between input images used in training versus those that result in failures during testing, (2) provide interpretable explanations of the subtle differences between those inputs, and (3) quantify the extent to which a dataset provides specification coverage. We also found that instantiating S^{3}C using an off-the-shelf scene graph generator on various open datasets illustrated its potential for use in practice, pinpointed the challenges that may arise, and identified the mechanisms to solve them.

ACKNOWLEDGEMENTS

We would like to thank Chris Morse for his invaluable help, insight, and prototyping at the outset of this work. We would also like to thank Carl Hildebrandt [18] for his invaluable help in implementing his artifact for use in this study. This work was supported in part by funds provided by National Science Foundation awards 2129824 and 2312487; and by The Air Force Office of Scientific Research under award number FA9550-21-0164. Trey Woodlief was supported by a University of Virginia SEAS Fellowship.

REFERENCES

[1] 2021. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. https://doi.org/10.4271/j3016_202104

[2] N. Amla and P. Ammann. 1992. Using Z specifications in category partition testing. In *COMPASS '92 Proceedings of the Seventh Annual Conference on Computer Assurance*. 3–10. https://doi.org/10.1109/CMPASS.1992.235766

[3] László Babai. 2016. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 684–697.

[4] Subho S Banerjee, Saurabh Jha, James Cyriac, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2018. Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 586–597.

[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] Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alexander G. Hauptmann. 2021. A Comprehensive Survey of Scene Graphs: Generation and Application. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), 1–1. https://doi.org/10.1109/TPAMI.2021.3137605

[7] Sully Chen. 2017. driving-datasets. https://github.com/SullyChen/driving-datasets

[8] 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)*.

[9] J. Deng, K. Li, M. Do, H. Su, and L. Fei-Fei. 2009. Construction and Analysis of a Large Scale Image Ontology. Vision Sciences Society.

[10] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. 2007. A Survey on Model-Based Testing Approaches: A Systematic Review *(WEASELTech '07)*. Association for Computing Machinery, New York, NY, USA, 31–36. https://doi.org/10.1145/1353673.1353681

[11] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. 2023. Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–48.

[12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[13] Jamil Fayyad, Mohammad A Jaradat, Dominique Gruyer, and Homayoun Najjaran. 2020. Deep learning sensor fusion for autonomous vehicle perception and localization: A review. *Sensors* 20, 15 (2020), 4220.

[14] Michael R Garey and David S Johnson. 2002. Computers and Intractability, vol. 29.

[15] Usman Manzo Gidado, Haruna Chiroma, Nahla Aljojo, Saidu Abubakar, Segun I Popoola, and Mohammed Ali Al-Garadi. 2020. A survey on deep learning for steering angle prediction in autonomous vehicles. *IEEE Access* 8 (2020), 163797–163817.

[16] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C Briand. 2020. Comparing offline and online testing of deep neural networks: An autonomous car case study. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 85–95.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[18] Carl Hildebrandt, Meriel von Stein, and Sebastian Elbaum. 2023. PhysCov: Physical Test Coverage for Autonomous Vehicles. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 449–461.

[19] Zhisheng Hu, Shengjian Guo, Zhenyu Zhong, and Kang Li. 2021. Coverage-based scene fuzzing for virtual autonomous driving testing. *arXiv preprint arXiv:2106.00873* (2021).

[20] Marko Ivanković, Goran Petrović, René Just, and Gordon Fraser. 2019. Code Coverage at Google. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 955–963. https://doi.org/10.1145/3338906.3340459

[21] 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.

[22] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. 2015. Image retrieval using scene graphs. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3668–3678. https://doi.org/10.1109/CVPR.2015.7298990

[23] Nidhi Kalra and Susan M Paddock. 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (2016), 182–193.

[24] Bogdan Korel, Inderdeep Singh, Luay Tahat, and Boris Vaysburg. 2003. Slicing of state-based models. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE, 34–43.

[25] Jiachen Li, Haiming Gang, Hengbo Ma, Masayoshi Tomizuka, and Chiho Choi. 2022. Important Object Identification with Semi-Supervised Learning for Autonomous Driving. (2022), 2913–2919.

[26] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. 2022. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *Knowledge and Information Systems* 64, 12 (2022), 3197–3234.

[27] 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

[28] István Majzik, Oszkár Semeráth, Csaba Hajdu, Kristóf Marussy, Zoltán Szatmári, Zoltán Micskei, András Vörös, Aren A Babikian, and Dániel Varró. 2019. Towards system-level testing with coverage guarantees for autonomous vehicles. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 89–94.

[29] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Harsimrat Kaeley, Anurag Karra, and Mohammad Abdullah Al Faruque. 2022. roadscene2vec: A tool for extracting and embedding road scene-graphs. *Knowledge-Based Systems* 242 (2022), 108245.

[30] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Deepan Muthirayan, Pramod P Khargonekar, and Mohammad Abdullah Al Faruque. 2022. Spatiotemporal Scene-Graph Embedding for Autonomous Vehicle Collision Prediction. *IEEE Internet of Things Journal* 9, 12 (2022), 9379–9388.

[31] T. J. Ostrand and M. J. Balcer. 1988. The Category-Partition Method for Specifying and Generating Fuctional Tests. *Commun. ACM* 31, 6 (jun 1988), 676–686. https://doi.org/10.1145/62959.62964

[32] Umar Ozgunalp, Rui Fan, Xiao Ai, and Naim Dahnoun. 2016. Multiple lane detection algorithm based on novel dense vanishing point estimation. *IEEE Transactions on Intelligent Transportation Systems* 18, 3 (2016), 621–632.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[34] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[35] A. Prakash, S. Debnath, J. Lafleche, E. Cameracci, G. State, S. Birchfield, and M. T. Law. 2021. Self-Supervised Real-to-Sim Scene Generation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 16024–16034. https://doi.org/10.1109/ICCV48922.2021.01574

[36] Ajitha Rajan, Michael Whalen, Matt Staats, and Mats P. E. Heimdahl. 2008. Requirements Coverage as an Adequacy Measure for Conformance Testing. In *Formal Methods and Software Engineering*, Shaoying Liu, Tom Maibaum, and Keijiro Araki (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 86–104.

[37] C Rodarius, J Duflis, F Fahrenkrog, C Roesener, A Varhelyi, R Fernandez, L Wang, P Seiniger, D Willemsen, and L van Rooij. 2015. Deliverable D7. 1: test and evaluation plan. (2015).

[38] Hajira Saleem, Faisal Riaz, Leonardo Mostarda, Muaz A Niazi, Ammar Rafiq, and Saqib Saeed. 2021. Steering angle prediction techniques for autonomous ground vehicles: a review. *IEEE Access* 9 (2021), 78567–78585.

[39] Eder Santana and George Hotz. 2016. Learning a driving simulator. *arXiv preprint arXiv:1608.01230* (2016).

[40] Artem Savkin, Rachid Ellouze, Nassir Navab, and Federico Tombari. 2021. Unsupervised Traffic Scene Generation with Synthetic 3D Scene Graphs. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1229–1235.

[41] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. Structural test coverage criteria for deep neural networks. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–23.

[42] Eric Thorn, Shawn C Kimmel, Michelle Chaka, Booz Allen Hamilton, et al. 2018. *A framework for automated driving system testable cases and scenarios*. Technical Report. United States. Department of Transportation. National Highway Traffic Safety Administration.

[43] 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.

[44] Udacity. 2016. self-driving-car. https://github.com/udacity/self-driving-car

[45] Mark Utting, Alexander Pretschner, and Bruno Legeard. 2012. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability* 22, 5 (2012), 297–312.

[46] Kelvin Wong, Yanlei Gu, and Shunsuke Kamijo. 2020. Mapping for autonomous driving: Opportunities and challenges. *IEEE Intelligent Transportation Systems Magazine* 13, 1 (2020), 91–106.

[47] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[48] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. Detectron2. https://github.com/facebookresearch/detectron2.

[49] Qian Yang, J Jenny Li, and David Weiss. 2006. A survey of coverage based testing tools. In *Proceedings of the 2006 international workshop on Automation of software test*. 99–103.

[50] Keren Ye and Adriana Kovashka. 2021. Linguistic structures as weak supervision for visual scene graph generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8289–8299.

[51] Jongmin Yu, Junsik Kim, Minkyung Kim, and Hyeontaek Oh. 2022. Camera-Tracklet-Aware Contrastive Learning for Unsupervised Vehicle Re-Identification. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 905–911.

[52] Shih-Yuan Yu, Arnav Vaibhav Malawade, Deepan Muthirayan, Pramod P Khargonekar, and Mohammad Abdullah Al Faruque. 2021. Scene-graph augmented data-driven risk assessment of autonomous vehicle decisions. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[53] Hong Zhu, Patrick A. V. Hall, and John H. R. May. 1997. Software Unit Test Coverage and Adequacy. *ACM Comput. Surv.* 29, 4 (dec 1997), 366–427. https://doi.org/10.1145/267580.267590

[54] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 79–90.