SD4AS: Safe Driving for Autonomous Systems

A Dissertation
Presented to
the Faculty of the School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment
of the requirements for the Degree
Doctor of Philosophy (Computer Science)

by

Felipe Toledo

December 2025

APPROVAL SHEET

This

Dissertation

is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Author: Felipe Toledo

This Dissertation has been read and approved by the examining committee:

Advisor:	Sebastian Elbaum
Committee Member:	Matthew B. Dwyer
Committee Member:	Ferdinando Fioretto
Committee Member:	Thomas Fletcher
Committee Member:	Corina Păsăreanu

Accepted for the School of Engineering and Applied Science:

[illegible]

Jennifer L. West, School of Engineering and Applied Science

December 2025

Acknowledgments

Completing a Ph.D. is not a task for only one person; it requires the support of many. I dedicate this section to the people who have contributed to and shaped my journey over the past few years. Space is limited, so I apologize for not explicitly mentioning everyone, but I still want to extend my gratitude.

First, I want to thank my advisor Sebastian for giving me the opportunity to be his student. Thank you for teaching me how to do research, think critically, and justify every single detail. Working with you has deeply shaped the way I think and see many things, not only in work but in different aspects of life. I will always be grateful for all the lessons I learned during my Ph.D.

I also want to extend my gratitude to my Ph.D. committee for their guidance. Corina, Tom, Nando, Matt, and Sebastian, thank you for taking the time to review my work and for your insightful questions. Your constructive comments and discussions throughout this process have significantly improved the quality of this dissertation.

Next, I want to thank the LessLab at the University of Virginia. To Matt, thank you for working together on almost all my research projects; it has been a pleasure to brainstorm ideas and discuss experiments with you. To David, thank you for all the help when I started the Ph.D. and your patience in answering a million questions about how everything works over Slack. To Carl, thank you for welcoming me to the lab, organizing board game nights, and doing science with the drones. To Will, thank you for our conversations in the lab and explaining so many things to me about American culture. To Trey, thank you for sharing your culture and improving my English writing and pronunciation. Also thank you for the coffee breaks, discussions, work on papers about scene graphs and autonomous vehicle safety, and bringing Cookout milkshakes before deadlines. And, to everybody in the LessLab, thank you for all the brainstormings, technical discussions, and support throughout all these years.

I also would like to thank my family. To my mom, Silvina, thank you for always taking care of me, from cooking to driving me everywhere when I was a kid. I am especially grateful that you forced me to go to my English classes when I did not want to; I would not be here today without that. Thank you for always reminding me to do my best and be happy. To my dad, Luis, thank you for long conversations on different matters; we will never run out of topics. Thank you for showing me different perspectives, convincing me not to give up when things get hard, and teaching me to keep pushing and not quit. To my sister, Josefina, thank you for reminding me to enjoy every day and not take life too seriously. And to my extended family, thank you for all the fun moments shared, and your unconditional support.

Last but not least, I would like to thank all my friends and girlfriend. To my friends in Argentina, thank you for all the good times, for receiving me with arms open every time that I come back, and for always believing in me. To my friends from Charlottesville, thank you for welcoming me here and sharing great moments. A special thanks to Ethan for sharing so much time together, teaching me the rules of different American sports, and helping me improve my English. To my girlfriend, Annie, thank you for supporting me all this time and adjusting to my busy schedule. Thank you for all the moments and activities we do together, including hiking, traveling, and cooking. Thank you for always having a smile on your face and making me happy.

I want to acknowledge the financial and technical support that enabled this research. I am grateful to the U.S. National Science Foundation for funding my work under grants 1900676, 2019239, 2129824, 2312487, 2332991, and 2403060. I also want to thank the University of Virginia for the support received through the Copenhaver Charitable Trust Bicentennial Fellowship. Beyond financial aid, I want to thank Research Computing at the University of Virginia (https://rc.virginia.edu) for providing the computational resources and technical support utilized in this dissertation.

Abstract

Autonomous Driving Systems (ADSs) are becoming increasingly widespread, with companies deploying them for various tasks such as taxi services, delivery, and personal transportation. As these systems become integral to daily life, ensuring their safe operation is crucial. However, recent high-profile safety incidents, including fatal collisions and traffic violations, have exposed the limitations of current validation approaches.

These failures stem from three fundamental problems that span the entire ADS lifecycle. First, there exists a semantic gap between the high-dimensional sensor data that ADSs consume—such as camera images and LiDAR point clouds—and the high-level safe driving properties against which they must be validated, preventing the formal specification and automated evaluation of safety requirements. Second, safety property violations are often discovered late in ADS development, requiring costly remediation efforts such as data collection and model retraining. Third, during deployment, ADS lack mechanisms for continuous monitoring and real-time correction to prevent safe driving properties violations and avoid potential accidents.

To address these problems, this dissertation introduces SD4AS (Safe Driving for Autonomous Systems), a framework designed to improve safe driving property conformance across the entire ADS lifecycle. SD4AS bridges the semantic gap by leveraging scene graphs—structured representations that capture entities and their spatial relationships—as an abstraction of raw sensor data. Combined with a specification language, this abstraction enables the definition and automated evaluation of complex safety properties, capturing 76% of the rules in the Virginia driving code. To improve conformance during development, SD4AS first quantifies the extent to which datasets contain scenarios necessary to validate properties, enabling developers to identify gaps in training data. Beyond data adequacy, for ADS that rely on machine learning components, SD4AS introduces property-aware optimization that treats safety rule violations as training errors, guiding the learning process to produce components that exhibit safe behaviors by construction, demonstrated by the fine-tuning two ADSs to reduce driving infractions. To maintain conformance during deployment, SD4AS synthesizes runtime monitors directly from property specifications, enabling continuous evaluation of safety rules during the ADS operation. To enable real-world monitoring, SD4AS integrates scene graph generators to extract entities and their spatial relationships in the driving domain directly from camera images. Lastly, SD4AS introduces a correction mechanism that proactively adjusts control outputs when violations are imminent, maintaining property conformance through real-time interventions, successfully reducing infractions of three different ADS architectures.

Through the development of these approaches and their empirical evaluation across multiple ADS, this dissertation advances the state of the art in ADS safety assurance. The contributions span the entire validation pipeline—from bridging sensor inputs and safety specifications, through improving conformance during development, to enabling continuous monitoring and correction during deployment—bringing us closer to safe autonomous vehicles.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous Driving Systems (ADSs) have rapidly moved from futuristic concepts to commercial reality, with autonomous vehicles now operating in major cities including Phoenix, San Francisco, Los Angeles, Austin, and Atlanta, with further expansion announced for Miami, Dallas, and Washington, D.C. by 2026 [1, 2]. ADSs are reshaping mobility with self-driving taxis [3, 4, 5], delivery vehicles [6], and personal transportation solutions [7]. The scale is substantial, with leading operators like Waymo providing over 250,000 paid trips weekly and traversing more than a million miles monthly [1, 2, 8]. This extensive operational volume increases the likelihood of ADSs encountering edge cases and complex traffic scenarios that can lead to accidents.

This expansion is occurring alongside conflicting safety data. On one hand, some operators report positive safety metrics. For instance, a 2025 Waymo study of its rider-only fleet, after 56.7 million miles, reported a significant reduction in injury-related crash rates compared to human benchmarks [9]. However, these findings are limited by the vehicles' highly restricted Operational Design Domains (ODDs). ADSs operate mainly on specific surface streets in select cities (like Phoenix and San Francisco) and exclude complex environments such as inter-state highways or severe weather. This makes a direct comparison to a general human benchmark difficult, as human drivers must navigate a far wider range of scenarios.

Moreover, these claims are overshadowed by severe, high-profile failures that highlight urgent safety needs, including incidents resulting in the loss of human life. For example, in one incident a Waymo vehicle crossed into the wrong lane and scraped the driver's side of another car before stopping a short distance down the road [10]. In another case, a car was rear-ended by a Waymo while stopping for a red light, knocking it into the center of the intersection [10]. Other notable incidents include a fatal Uber test vehicle collision with a pedestrian [11, 12] and a Cruise robotaxi striking and dragging a pedestrian, which triggered intense regulatory scrutiny [13]. Beyond these events, there has been repeated instances of AVs failing to react to stationary hazards and emergency vehicles, leading to collisions [14, 15, 16]. The spectrum of failures also includes fundamental traffic violations like making u-turns or running red lights [17]. The consequences extend beyond immediate harm. The Cruise incident serves as a stark example: following the event, the company's permit was suspended, leading to a nationwide service halt and eventual losses for its parent company, General Motors, exceeding $10 billion [13]. This demonstrates that safety is not merely a technical or ethical consideration but a primary determinant of corporate viability.

These high-profile failures have fueled a crisis of public trust. Recent studies by the American Automobile Association (AAA), a long-standing federation of motor clubs that serves as a leading consumer advocate for automotive safety, show that a significant majority of U.S. drivers—approximately six in ten—report being "afraid" to ride in a self-driving vehicle [18, 19]. Similarly, a poll by YouGov, an international market research and data analytics group, found that 70% of Americans are worried about the widespread use of driverless cars, with a majority actively opposing their operation in their own cities [20]. In parallel, governmental bodies are escalating oversight. At the federal level, the National Highway Traffic Safety Administration (NHTSA) now requires biannual reports on AV research, mandates the reporting of all crashes involving automated systems, and has proposed new frameworks like the AV STEP program to increase transparency [21, 22, 23]. At the state level, legislation like California's Assembly Bill No. 1777 is closing accountability loopholes by making AV operators legally liable for moving violations [17]. This feedback loop—where incidents fuel public skepticism and trigger regulatory action—makes proactive safety a critical strategy for de-risking future deployments.

The confluence of rapid deployment, the high consequence of failure, fragile public trust, and escalating regulatory scrutiny creates an urgent need for a new framework that can improve ADS compliance with safe driving properties. This dissertation argues that improving ADS compliance to safe driving rules requires solving three main problems. The following section details these problems and the solutions proposed to address them.

1.1 Problems for Improving ADS Safety

Improving the safety of ADSs requires addressing three interconnected problems that contribute to failures that are eroding public trust and triggering regulatory action. First, it is currently not possible to formally specify and validate safe driving properties over sensor data, preventing the ADS safety validation. Second, insufficient safety integration during development results in costly late-stage failures like those experienced by Cruise and Uber. Third, the absence of runtime monitoring and enforcement leaves deployed systems vulnerable to safety violations. These problems span the entire ADS lifecycle—from specification through development to deployment—and addressing them is a significant step toward safer AVs.

The first problem is validating ADSs that consume high-dimensional sensor data, such as camera images or LiDAR point clouds, against high-level safe driving properties. This is a non-trivial task due to the disconnect between raw data inputs and safety requirements The complexity of sensor data, represented by raw pixels and point clouds, makes it difficult to reason about properties. For example, validating a property like *"If there is a concrete barrier to the left, the ADS should not steer to the left"*, would require determining the presence and location of a concrete barrier in images and point clouds. However, even when equipped with a perception model that can provide such information, there is still a disconnect between the perception output and the property's requirements. Perception models output geometric data, such as 3D bounding boxes, but the safety property requires inputs with matching semantics, such as a boolean predicate `barrierToLeftOf`. On the other hand, evaluating semantic terms like something is "to the left" involves ambiguous design choices about an object's position or proximity, each affecting the property's validation. Thus, the lack of a mapping from low-level sensor data to high-level semantic concepts prevents the specification and validation of safe driving properties.

The second problem is that safety property violations are often discovered late in the system development process. Failures detected during validation must be corrected before deployment, which can require costly interventions such as collecting additional data and retraining complex Neural Network (NN) components. The necessity for additional data arises because collecting large volumes of inputs does not necessarily covers rare, semantically distinct scenarios, leaving gaps in the dataset. Likewise, the standard optimization processes maximize accuracy rather than explicit adherence to safety rules, and can make models to internalize unsafe behaviors despite high performance metrics. Identifying these deficiencies in datasets and trained NN after the ADS is built increases the computational and logistical costs of the development cycle.

The third problem is ensuring that ADSs continue to satisfy safe driving properties after deployment to avoid catastrophic outcomes, including crashes. Distinct from the development phase, violations in this stage arise when the system encounters novel scenarios that were not present in the training data. Failures that occur during operation, require remediation at runtime rather than offline retraining. This constraint highlights the need for an approach capable of interpreting sensor data to monitor property conformance and managing the ADS's behavior to intervene when safety violations are imminent.

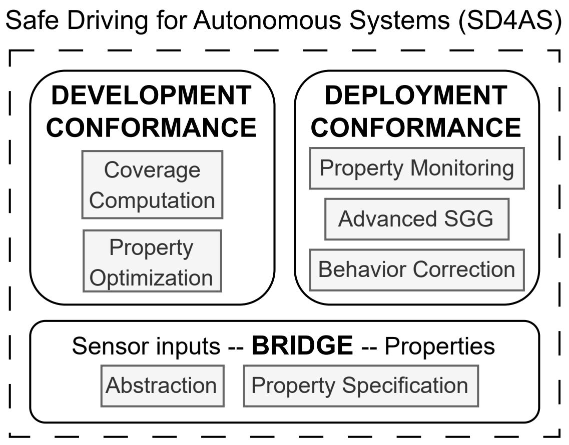


Figure 1.1: The SD4AS framework conceptual diagram.

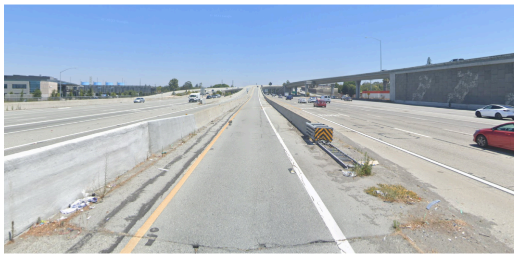
To address these problems, we propose SD4AS (Safe Driving for Autonomous Systems), a framework shown in Figure 1.1, designed to improve safe driving property conformance in ADSs. SD4AS does so by (1) establishing a **Bridge** between high-dimensional sensor inputs and safe driving properties, that uses Scene Graphs (SGs) to abstract the sensor inputs, enabling the specification and evaluation of properties; (2) improving **Development Conformance** by quantifying dataset coverage of semantically distinct scenarios and explicitly enforcing property adherence during the model's optimization process; and (3) maintaining **Deployment Conformance** by advancing SGG to enable real-time monitoring of safe driving properties during deployment and correcting unsafe behaviors at runtime.

1.1.1 Bridging Sensor Inputs and Safe Driving Properties

Bridging the gap between sensor inputs and safe driving properties to enable their evaluation requires overcoming two challenges.

The first challenge is to find a suitable abstraction to reason about the high-dimensional sensor inputs. After defining a criteria and investigating multiple options, we selected SGs as the preferred abstraction due to their ability to capture relational and spatial information relevant to ADS decision-making. SGs structure complex data into interpretable entities with attributes and their relationships, making them suitable for specifying and evaluating safe driving properties because these properties typically define spatial interactions between objects, such as a vehicle maintaining a safe distance from a car ahead. This utility is supported by the effective use of SGs in domains like Visual Question Answering (VQA) [24] and 3D scene understanding [25, 26], where SGs enable systems to query visual scenes for specific semantic relationships, similar to evaluating safety predicates in ADS environments.

The second challenge is to enable the definition of safe driving properties over the selected abstraction. Consider an input image captured from an ADS's camera showing a concrete barrier, its extracted SG (as shown in Figure 1.2), and the previously mentioned safety rule that states: *"If there is a concrete barrier to the left, the ADS should not steer to the left"* Formalizing such property requires a language capable of specifying it precisely and querying the SG to identify specific entities



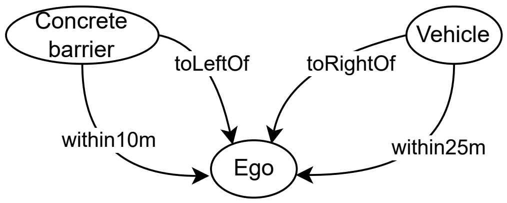


Figure 1.2: Input image and its SG.

and their spatial relationships. In this case, we would need to check for a `concrete barrier` node connected by a `toLeftOf` edge to the `ego` vehicle. Evaluation is then performed by querying the generated SG to determine if this precondition exists; if it does, we then check that the ADS's control output satisfies the constraint of not steering left. To address this challenge, we designed and integrated a Domain-Specific Language (DSL) into SD4AS to specify and validate safe driving properties using SGs. This DSL can operate in conjunction with Linear Temporal Logic over Finite Traces LTL_f, which we leveraged within SD4AS to specify and validate more complex temporal behaviors over bounded scenarios, critical for autonomous systems.

1.1.2 Increasing property conformance in ADS development

Building on the SG abstraction, SD4AS improves conformance with safe driving properties early in the development process. This is important because waiting to correct safety violations found during validation or deployment can be costly, often requiring data collection and retraining of NN components. Proactively embedding safety into the development lifecycle requires overcoming two challenges to enable that ADSs begin learning safe driving behaviors as they are constructed.

The first challenge is ensuring that the dataset used for training the NN component adequately covers the range of conditions the ADS will encounter. For instance, if an ADS has not been trained with scenes involving concrete barriers, it is likely going to be unprepared to handle these situations safely in deployment. To address this, SD4AS assesses data adequacy by quantifying the coverage of scenarios relevant to specified safe driving properties. By mapping sensor data to SGs, the framework measures the dataset diversity and the frequency of critical spatial scenarios relevant for the specified properties. This process provides an assessment of whether the ADS has access to the necessary data to learn safe behaviors.

The second challenge is that exposing the system training to relevant data alone does not guarantee conformance with safe behavior. ADSs must not only encounter these scenarios but also consistently recognize and avoid unsafe actions within them. For example, in the same scene with a concrete barrier on the left, the ADS must learn that steering left is unacceptable. Accordingly, the training process must explicitly penalize such predictions to reinforce safe behaviors. To achieve this, SD4AS uses SG to abstract sensor inputs to evaluate property conformance and augments the dataset to have sufficient examples of safety-critical scenarios. It then integrates a custom loss function and Lagrangian optimization into the training pipeline to penalize safe driving property violations, promoting safer decision-making by actively discouraging unsafe actions across varied scenarios

1.1.3 Monitoring and improving property conformance in ADS deployment

Ensuring safe ADS operation after deployment requires overcoming three challenges that arise in real-world environments. This includes the need for continuous monitoring of safety properties, accurate interpretation of complex sensor data, and the ability to intervene when violations are detected.

The first challenge is to check that the safe driving properties hold not only during the ADS validation phase but also in real-time during its deployment. While validation ensures that properties are well-defined and respected under controlled settings, deployment introduces unpredictable scenarios, making continuous monitoring essential to maintain safety. Building on the formal language to specify properties and the SG abstraction, SD4AS incorporates an approach that leverages them to enable continuous evaluation during deployment. This approach synthesizes runtime monitors directly from the specified properties, which then translates sensor inputs into SGs, evaluate the corresponding safety rules, and track their status over time to systematically detect and report violations.

The second challenge arises from the limitations of current Scene Graph Generators (SGGs) for autonomous driving, including insufficient accuracy and limited ability to capture the complexity of real-world driving scenarios. To enable SD4AS's monitoring capabilities, we initially used accurate SGs extracted from simulation data, which provided a reliable foundation for development and evaluation. However, for practical deployment in real-world settings, it is necessary to generate SGs directly from real images and a wide variety of camera sensors, rather than relying on simulation. To address this challenge, we explored a family of Visual Language Models (VLMs) to extract SG from the driving domain. We also fine-tuned VLMs for this application, improving SGG accuracy and enabling monitoring of safe driving properties beyond simulation environments.

The third challenge is enabling proactive correction of ADS behavior during deployment to reduce safe driving property violations in the field. Although training optimizations through techniques like T4PC can improve adherence to safety rules, it does not guarantee that the ADS is not going to fail during operation due to the stochastic nature of NN and the exposure to new scenarios during deployment. For example, if the monitor detects that the ADS is about to steer toward a concrete barrier, the corrector mechanism can override the steering signal to prevent a collision. To address this, SD4AS incorporates a corrector mechanism that can adjust the ADS's output signals—such as steering or acceleration—when a violation is imminent. This approach allows SD4AS to intervene the ADS in real time and trigger safe actions, helping to prevent collisions and maintain conformance with safe driving rules.

1.2 Contributions

This dissertation introduces SD4AS, a framework designed to improve safe driving property conformance for ADSs that process high-dimensional sensor inputs. By using the DSL and SGs, SD4AS enables the specification and evaluation of these properties. Furthermore, it improves conformance during ADS development by quantifying dataset coverage and enforcing property conformance during the training process. Separately, it improves safety during deployment by monitoring for violations in real-time and intervening to correct unsafe behaviors. Table 1.1 summarizes the three problems this dissertation focuses on, their challenges, the SD4AS's contributions to overcome them, and the publications and artifacts disseminating this contribution to the broader research community.

Problems	**Challenges**	**Contributions**	**Papers**
Bridging Sensor Inputs and Safe Driving Properties (Chapter 3)	Selecting and Building an Abstraction for Sensor Data (Section 3.1)	• Selection of SG as abstraction and tailoring it to the ADS domain • Design and implementation of CARLA SGG plugin	ICSE 25 Demo [27] [A1] [1]
	Specifying Safe Driving Properties (Section 3.2)	• Design, implementation and assessment of DSL to query SG and define LTL_f properties	ICRA 24 [28] [A2] [2] FSE 25 [29] [A3] [1] [3]
Increasing Property Conformance in ADS Development (Chapter 4)	Computing Dataset Property Coverage (Section 4.1)	• Approach and implementation to compute dataset property coverage, and its evaluation	ICSE 24 [30] [A4] [2]
	Integrating Properties in ADS Development (Section 4.2)	• Approach and implementation to train NN for property conformance, and its evaluation	TSE [31] [A5] [4]
Monitoring and Improving Property Conformance in ADS Deployment (Chapter 5)	Monitoring Safe Driving Properties (Section 5.1)	• Approach and implementation for synthesizing runtime monitors, and its evaluation	ICRA 24 [28] [A2] [2]
	Advancing Scene Graph Generation for Real-World Deployment (Section 5.2)	• Approach and implementation to extract SG from 3D datasets • Family of VLMs for generating SG	ERAS 25 [32] [A6]
	Correcting ADS behavior (Section 5.3)	• Approach and implementation to correct ADS outputs to ensure conformance, and its evaluation	ICRA 26 [33] (Submitted)

Table 1.1: Summary of challenges and contributions.

[1] In collaboration with Trey Woodlief. Minor contribution.
[2] In collaboration with Trey Woodlief. Equal contribution.
[3] Not included in dissertation.
[4] In collaboration with Trey Woodlief. Major contribution.

Chapter 2

Background

The promise of ADS to reduce traffic accidents and improve mobility relies on the assurance of safety. However, ensuring this safety requires overcoming the semantic gap between the high-dimensional, unstructured sensor data that ADS consume—such as camera images and LiDAR point clouds—and the high-level, safe driving properties against which they must be validated. Current ADS validation approaches are unable to bridge this gap, as they lack both the mechanisms to abstract raw sensor data into interpretable representations and the formalisms required to specify safe driving rules over these abstractions. The problem is exacerbated by the architectural transition from traditional software to systems driven by Artificial Intelligence (AI) and Machine Learning (ML) components. As articulated by Koopman and Widen[34], this new kind of system introduces a distinct set of risks, as the deployment of these technologies faces the "general brittleness of machine learning technology to novel situations" where shifts in the operational environment or rare long-tail events can lead to unpredictable failures. The unpredictability of ML components makes bridging the semantic gap even more critical; as the nature of these components prevents developers from reasoning directly about their behavior, the formal specification and systematic evaluation of safety properties become an important strategy for assuring safety.

This semantic gap — and its amplification by ML components — manifests as a set of safety challenges across the ADS lifecycle. During development, safety property violations are often discovered late, requiring costly remediation such as data collection and model retraining. This occurs because training datasets, despite their size, often lack adequate coverage of safety-critical scenarios, preventing the model from learning how to handle these specific situations. Furthermore, ML components are optimized for accuracy, meaning that even models with high performance metrics may fail to internalize and adhere to safe driving rules. During deployment, the unpredictability of ML components necessitates mechanisms to continuously monitor that the ADS behavior conforms to safe driving properties and to correct the ADS in real-time to prevent violations. Therefore, establishing that an AV is safe enough requires a framework that can bridge the semantic gap to specify properties, while simultaneously addressing the nondeterministic nature of ML components through rigorous development conformance and runtime enforcement.

To address these problems, this dissertation explores approaches to specify and validate safe driving properties, improve conformance of the ML components to these properties during development, and monitor and correct property violations during the deployment of the ADS. This chapter lays the background for these contributions. Section 2.1 reviews perception techniques, specifically focusing on Scene Graph Generators and Visual Language Models as means to extract structured semantic representations from raw sensor data, providing the abstraction necessary to bridge the semantic gap between high-dimensional sensor inputs and high-level safety properties. Section 2.2 introduces property specifications, which leverage these abstractions to enable the formal definition and automated evaluation of safe driving requirements. More specifically, it details Relational First-Order Logic and Linear Temporal Logic (RFOL) used to formally define safety constraints and Linear Temporal Logic (LTL) used to formally define temporal aspects. Finally, Section 2.3 discusses property compliance, examining existing methods in data coverage, runtime monitoring, and informed machine learning that aim to ensure AV behaviors adhere to these specified safety properties, addressing the problems of conformance during both development and deployment.

2.1 Perception

Perception is the process of interpreting sensory inputs to understand the physical world [35]. Its function is to transform raw sensor data, such as camera images or LiDAR point clouds, into a meaningful scene representation such as scene graphs [36]. This transformation is necessary because raw sensor inputs are high-dimensional and lack the explicit semantic structure required to specify and evaluate safe driving rules. To understand the world, this dissertation uses two complementary approaches to extract scene representations from raw sensor data. We investigate two approaches for generating these graphs. Scene graphs generators provide a structured, graph-based representation that captures the spatial and semantic relationships between entities in a scene. Visual Language Models (VLMs) represent an alternative approach that leverage learned semantic relationships from multimodal training data to produce these structured graphs. Both approaches are capable of producing scene representations that can be used to evaluate formal property specifications, which we discuss in Chapter 3.

2.1.1 Scene Graph Generator

A Scene graphs Generator (SGG) is a method that can automatically extract Scene Graph (SG) which are structured representations that provide an intuitive way to represent the world by capturing what objects are present and how they relate to each other. For example, consider the driving scenario shown in Figure 2.1, where a camera observes a traffic intersection from the ego vehicle's perspective A SG represents this scene as a graph where nodes capture the entities (ego vehicle, other cars, lanes, traffic light) with their attributes (e.g., the traffic light is red), and edges capture their spatial and semantic relationships [36, 37]. As illustrated in the figure, the ego vehicle "isIn" the ego lane, one car "isIn" ego lane and the other is in the right lane, the right lane is "toRightOf" the ego lane, and a red traffic light "controlsTrafficOf" the ego lane. This structured representation makes it natural to reason about complex scenarios: we can query whether there exists a red traffic light controlling our lane, determine which vehicles are in adjacent lanes, or assess potential collision risks.

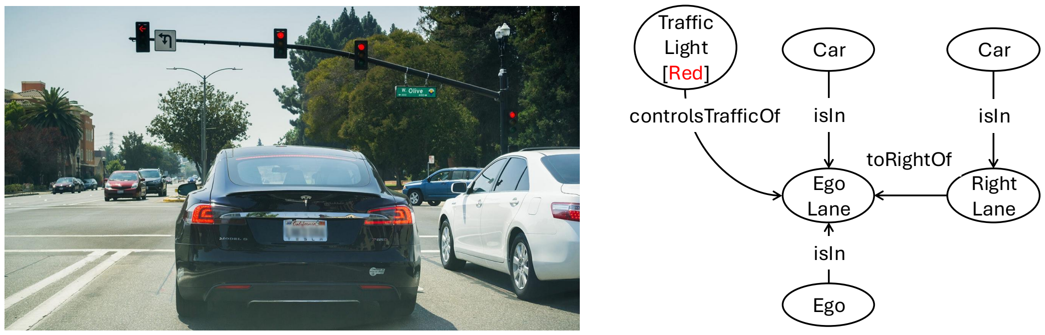


Figure 2.1: Example scene graph representation of a driving scenario showing entities (nodes) and their relationships (edges). The traffic light's red state is captured as an attribute.

The SGG process has traditionally been categorized as either *bottom-up* or *top-down*. Bottom-up approaches require the initial identification of objects and their attributes, typically through an object detection network like Yolo [38] or Detectron [39], followed by the identification of the relationships between the detected objects [40, 41, 42, 43]. Conversely, top-down approaches aim to detect and recognize objects and their relationships simultaneously [44, 45, 46, 47]. Building on these paradigms, recent SGG models adopted end-to-end transformers that jointly predict objects and relations as a sparse set of triplets or graph structures [48, 49, 50, 51], which improves efficiency and mean-recall on benchmarks such as Visual Genome [52] and PSG [53]. Beyond static images, more sophisticated SGGs extend these capabilities to the temporal domain, tracking objects across video frames [54] and generating relations from natural language captions [55]. Dynamic and video SGG methods further advance this by explicitly modeling temporal consistency and anticipating future relations, for example through anticipatory pre-training [56], hierarchical aggregation with cyclic temporal refinement to maintain long-range coherence [57], or unified hypergraphs that model higher-order interactions and causal transitions for reasoning and anticipation [58].

In terms of application, SGGs range from generic models [53, 59] trained on general-purpose datasets to those specialized for specific domains. These include image generation [60, 61] and image captioning [62], where structured semantic information represented by an SG improves performance. Other applications include Visual Question and Answering (VQA) [24], where SGs capture the essential information of images to enable reasoning over defined objects and relationships, as well as

3D scene understanding [25, 26], which aims to construct 3D SGs by incorporating accurate spatial relationships in 3D space.

Within AVs, the need for task-specific scene understanding has motivated the development of specialized SGGs tailored to driving scenarios [63, 64, 65]. Recent work in this direction explore ego-centric traffic scene graph benchmarks [66] and road scene graph models that integrate HD-map priors and traffic semantics [67], consistent traffic scene graphs generation through constraint optimization [68], and robust spatial–temporal scene graph construction under perception failures [69]. These domain-tailored SGGs leverage vehicle-specific semantics—such as road types, vehicle classes, and the distinction between static and dynamic entities—to generate more informative scene representations than generic SGGs. A notable example is RoadScene2Vec [70], an open-source toolset for generating, encoding, and utilizing road SGs. RoadScene2Vec has the ability to generate SGs from single images. The generator offers a rich set of configuration parameters that facilitate the experimentation of different pixel to SG abstractions. For example, the configuration file includes parameters that define what entities to include (car, pedestrian, etc.), what position relations 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 distance relations (e.g., near collision, super near, very near, near, and visible) can be instantiated also with their thresholds (e.g., [0, 4), [4, 7), [7, 10), [10, 16), and [16, 25) meters). Despite RoadScene2Vec being highly configurable, it struggles to identify the correct spatial relationships between objects, leading to low precision and the creation of inaccurate SGs.

2.1.2 Visual Language Models

Visual Language Models (VLMs) [71] possess a joint understanding of both visual and textual information enabling multiple vision-language tasks such as image captioning [72, 73], visual question answering (VQA) [74], image classification [75], and object detection [76]. This rapidly evolving field has produced a diverse array of models that push the boundaries of multimodal performance. Among these, GPT-5 [77], Gemini 3 [78], and Claude 4 [79] serve as benchmarks for top-tier multimodal capabilities, showcasing exceptional performance across a variety of tasks. Foundational models such as LLaVA 1.5 [80] exemplify the synergy between robust visual encoders and advanced language models. LLaVA 1.5 integrates the widely used CLIP [75] image encoder with Llama 2 [81], creating a powerful model for vision-language tasks. Building on this foundation, LLaVA 1.6 [82] introduces two variants—Mistral and Vicuna—designed to handle higher image resolutions and fine-tuned on a more diverse dataset. These models leverage the latest advancements in their respective language models, providing enhanced performance and adaptability to complex tasks. Further expanding these capabilities, DeepSeek-VL2 [83] introduces a Mixture-of-Experts architecture with dynamic tiling for efficient high-resolution processing, and Qwen2.5-VL [84] extends model capabilities beyond passive analysis, enabling them to comprehend long videos and actively operate computer and phone interfaces.

Despite these advances, a key limitation of general-purpose VLMs is their weak spatial reasoning capabilities [85], which is critical for safety-sensitive applications like AVs. Recognizing this need, recent works have developed spatially-aware VLMs that enhance spatial understanding through architectural innovations. Spatial VLM [86] proposes a pipeline to automatically generate spatial labeled data to improve VLM's capacity for spatial reasoning, while the Cambrian [85] family of models (including Phi 3 [87] and Llama 3 [88]) incorporates the Spatial Vision Aggregator (SVA), which aggregates features from multiple vision encoders to enable more nuanced spatial reasoning. In autonomous driving, this advancement is particularly important since AV properties often rely on precise distance and spatial relationships between entities; for example, determining whether an ego vehicle should brake requires knowing the distance to an obstacle ahead. Building on these advancements, Nvidia's CubeLLM [89] addresses the need for spatially-aware datasets by introducing LV3D, a large, diverse, and spatially-annotated dataset that includes numerous driving datasets with 3D labels. CubeLLM leverages LV3D to train a VLM for multi-turn question answering, demonstrating its potential for handling complex, spatially-grounded tasks in autonomous driving scenarios.

Recent works investigate scene graph generators using open vocabulary [90, 91, 92]. Notably, the work in [92] employs a VLM to transform images into a sequence of relation-aware tokens and convert them into structured scene graphs. This integration of VLMs with scene graphs combines the semantic understanding capabilities of VLMs with the structured, queryable nature of SGs, enabling richer and more interpretable scene understanding for downstream tasks. However, its reliance on 2D bounding boxes restricts the representation to the image plane, failing to capture the 3D spatial distances and domain-specific relations necessary for autonomous driving.

2.2 Property specification

Specifying properties require a formal languages that can precisely express complex scenarios and system requirements. This section presents two complementary formal approaches for specifying properties for AV systems. Relational First-Order Logic (RFOL) provides the foundation for expressing static properties over structured relational data, enabling precise descriptions of the spatial and semantic relationships between entities in a scene. Linear Temporal Logic (LTL) [93] extends this capability to the temporal domain, allowing specifications of how these relationships and system states must evolve over time. Together, RFOL and LTL form the basis for defining safety-critical properties used for validating the behavior of AVs.

2.2.1 Relational First Order Logic

Relational First-Order Logic (RFOL) is a formal language used to express properties over relational structures, which are essentially sets of objects and the relationships between them. RFOL provides a precise, powerful, and unambiguous syntax for defining properties. This makes it ideal for specifying complex scenarios, requirements, or queries in various domains where structured information is important. An RFOL formula is constructed from a vocabulary that defines the available types of objects and relationships, combined with variables, logical connectives, and quantifiers.

A RFOL formula ϕ is built with the following components. First, it uses variables (e.g., $x, y, z, \dots$) that represent arbitrary objects within the domain being discussed. Second, it employs predicates, which are symbols representing properties of objects or relationships between them. Unary predicates describe properties of single objects (e.g., $\texttt{Car}(x)$ meaning 'x is a car', or $\texttt{isRed}(x)$ meaning 'x has the property red'), while binary predicates describe relationships between two objects (e.g., $\texttt{isIn}(x, y)$ meaning 'x is in y', or $\texttt{toRightOf}(x, y)$ meaning 'x is to the right of y'). Third, standard logical connectives like $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\rightarrow$ (implies) are used to combine simpler formulas into more complex ones.

The meaning (semantics) of an RFOL formula ϕ is determined by evaluating it against a specific model or structure, M. This model M contains the actual objects, their attributes, and the relations between them. We say that the model M satisfies the formula ϕ, written $M \models \phi$, if the formula is true according to the objects and relationships defined in M. The truth of a formula is defined recursively. For example, $M \models \texttt{Car}(x)$ is true if the object assigned to the variable x in M is indeed a car. Similarly, $M \models \texttt{isIn}(x, y)$ is true if the relationship "isIn" holds from the object assigned to x to the object assigned to y within M. Compound formulas using connectives like $\wedge$ are true if both sub-formulas are true (e.g., $M \models \phi_1 \wedge \phi_2$ if $M \models \phi_1$ and $M \models \phi_2$), with similar rules for $\vee$, $\neg$, and $\rightarrow$.

2.2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) [93] is a formal language designed for specifying and verifying the behavior of systems that evolve over time, e.g. embedded or cyber-physical systems [94, 95, 96]. It enables the definition of temporal relationships between events or states, using different operators to capture the progression of time. LTL operates over traces of Boolean values encoding the semantics of the events or states. The logical operators include: *And ($\wedge$)*, *Or ($\vee$)*, *Not ($\neg$)*, among others, while the temporal operators are: *Next ($\mathcal{X}$)*, *Until ($\mathcal{U}$)*, *Always ($\mathcal{G}$)*, and *Eventually ($\mathcal{F}$)*. For example, the LTL property $\mathcal{G}((\text{car\_within\_4m} \wedge \text{car\_in\_front}) \rightarrow \mathcal{X}\ \text{brake\_engaged})$ specifies that whenever there is a car within 4 meters and in front of the ego vehicle, the brakes must be engaged in the next state, and this rule must hold at all times during system execution. Another example is $\mathcal{G}((\text{red\_traffic\_light} \wedge \text{ego\_approaching}) \rightarrow \mathcal{X}\text{brake\_engaged})$, which specifies that whenever the ego vehicle approaches a red traffic light, the brakes must be engaged in the next state.

Linear Temporal Logic over Finite traces (LTL$_f$) [97] is an extension of traditional LTL specifically tailored for tasks that operate over finite time horizons, such as discrete tasks that have a clear start and end point. Its ability to express temporal relationships over finite traces can encode finite temporal events like passing a vehicle or changing lanes, where correctness is tied to specific sequences of events. A property expressed in LTL$_f$ can be transformed into a Deterministic Finite Automaton (DFA) [98, 99], which efficiently checks whether a finite trace satisfies or violates the property, making it well-suited for real-world applications in task planning and rule adherence. A DFA starts from a specific initial state determined by the LTL$_f$ formula and then transitions through states based on the semantics of the formula. A trace ending in a non-accepting state represents a violation of the property, while a trace ending in an accepting state satisfies the property.

2.3 Property Compliance

This section reviews prior work related to the challenge of ensuring an ADS adheres to specified safety properties and driving rules. We structure this review around three complementary research areas. First, in Section 2.3.1, we discuss Coverage, which focuses on techniques for quantifying the extent to which a system's compliance has been adequately tested across diverse scenarios. Second, in Section 2.3.2, we explore Runtime Monitoring and Enforcement, which focuses on techniques that actively check for property violations during operation and, in some cases, intervene to prevent them. Finally, in Section 2.3.3, we examine Informed Machine Learning, which focuses on techniques for building inherently compliant components by integrating property constraints and domain knowledge directly into the model training pipeline.

2.3.1 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 intepretability, discrimination, and automation. Automation refers to the capacity to compute the coverage metrics without manual intervention. Interpretability ensures that coverage metrics can be understood by humans. Finally, discrimination ensures the coverage metric effectively distinguishes between semantically unique cases, allowing developers to pinpoint specific coverage gaps.

White-box structural code coverage metrics have been widely studied and adopted for their ease of use and basic interpretability for developers [100, 101, 102]. 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 [103]. 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 [104]. Prior work generalized white-box structural coverage to neural networks [105, 106, 107]. Such coverage can be computed *automatically*, but it is not *interpretable* due to the opaqueness of neural networks [108].

Black-box coverage includes a range of approaches from those focused on the domain derived directly from the requirements [109, 110, 111] to those on domains defined by system and environmental models [112, 113]. 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 [114] or (b) machine-learned models of the input space [115]. 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.* This restriction arises because the input model requires an initial human labor to identify meaningful abstractions that cannot be derived algorithmically. For instance, DeepHyperion [114], relies on a manual procedure where human experts must inspect and label dataset samples to define relevant feature dimensions, such as the boldness of a handwritten digit or the complexity of a road. 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* [108].

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 *discriminatory* or *interpretable* [116, 117, 118]. Majzik et al. defined scenario coverage [119], 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* [120, 121]. Hu et al. compute trajectory coverage by examining utilization of road regions within a scenario [122]. It has limited *discrimination* power as it makes over-simplifying assumptions about the road structure, and ignores entities. Closest to meeting the three requirements is Hildebrandt et al.'s recent work PhysCov [104] 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*, 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*, *discriminatory*, and *automated* coverage metric for the AV domain at the input level.

2.3.2 Runtime Monitoring and Enforcement

Prior work has developed monitors for specific AV subcomponents, such as adaptive cruise control [123], collision avoidance [124, 125, 126], trajectory prediction [127], or lane changing and overtaking [128, 129]. In contrast, other works have focused on monitoring end-to-end AV systems. These approaches often use formalisms like Signal Temporal Logic (STL) [130, 123, 131, 132], Linear Temporal Logic (LTL) [133, 134], or First Order Logic (FOL) [135]. Among these, two particularly relevant works address the challenge of reasoning about complex spatial relationships between entities. Morse et al. [135] leverage graph representations to abstract system traces and unsupervisedly infer relations between sensed object distributions and robot behaviors. However, this approach focuses on the discovery of behavioral specifications rather than the active monitoring and enforcement of explicit safety requirements. In contrast, Pedro et al. [134] propose a method for monitoring spatio-temporal properties defined over metric spaces. While effective for checking geometric constraints over time, this approach lacks support for semantic constraints—such as detecting a "red" traffic light—which are essential for validating conformance with high-level driving rules.

Building upon monitoring, some research focuses on the offline synthesis of enforcement mechanisms. In these approaches, safety constraints are integrated during the development phase, either by training the system to internalize these properties or by generating a repair artifact to guide the vehicle during deployment. This typically involves synthesizing compliant strategies or repair scripts before operation. For instance, approaches using reinforcement learning shielding can learn to satisfy properties defined in temporal logic without active runtime intervention [136, 137, 138, 139]. However, these methods typically operate within a Markov Decision Process (MDP) framework to shield unsafe actions based on a known state. Consequently, they are difficult to apply in end-to-end deep learning settings, where the complex mapping from raw sensor data to these abstract states is often undefined. On the data-driven side, FixDrive proposes an offline framework that analyzes driving records from previously recorded violations [140]. It uses a Multimodal Large Language Model (MLLM) to reason over "critical moments" from a record, such as near-misses. The MLLM then automatically generates a repair script in a high-level domain-specific language. This generated script is subsequently applied to the ADS to dynamically adjust its driving strategy at runtime. As this is a data-driven synthesis approach, its primary limitation is that it can only generate rules for violations that the vehicle has already encountered and recorded. ADReFT [141] also offers a repair mechanism that learns offline from a corpus of failed tests. It uses a transformer-based model trained with supervised learning on weakly annotated violation data and refined using reinforcement learning. Like other learning-based methods, its core limitation is that the trained model will only learn to recognize and repair situations that are sufficiently close to the violation data used during its training.

In contrast to offline synthesis, online enforcement techniques integrate monitoring directly with the system's decision-making to prevent violations in real-time. For example, online shielding mechanisms enforce safety properties by overriding unsafe actions during operation [142]. However, this real-time validation typically requires computing future trajectories within a known state space, which is often infeasible for end-to-end deep learning systems that must process raw sensor inputs without a clear mapping to these abstract states. Alternatively, REDriver [132] introduces a modular framework for runtime enforcement positioned between an ADS's motion planning and control modules. It allows developers to specify safe driving properties, such as national traffic laws, using STL. At runtime, REDriver evaluates the ADS's planned trajectory against these properties using a robustness metric. If this score indicates a likely violation, it employs a gradient-driven algorithm to compute a minimal repair to the trajectory, which is then passed to the control module. A significant limitation of this approach, however, is it requires special instrumentation of the ADS to extract the semantic predicates and it assumes that the perception data received by the ADS perfectly reflects the physical environment.

A key challenge for the logic-based enforcement methods described earlier, such as FixDrive and REDriver, is that they take for granted the evaluation of their atomic propositions (APs). They often ignore the critical mapping required between raw sensor inputs and the high-level AP truth values. To begin addressing this gap, Anderson et al. [143] introduced spatial regular expressions to match patterns over a sequence of images. This work, however, is limited by reasoning only about 2D bounding boxes, which over-approximate the shape of objects and are imprecise for 3D reasoning.

2.3.3 Informed Machine Learning

Purely data-driven approaches to learning may perform poorly when there is insufficient data to train a generalized NN, or when the NN must adhere to constraints imposed by natural laws, regulations, or guidelines [144]. The field of *informed machine learning* investigates the integration of such prior knowledge into the training pipeline [145]. This prior knowledge can come from various sources, such as scientific knowledge, which formalizes principles from physics or engineering, or world knowledge, which encompasses everyday facts that are less formal—like the fact that a dog has four legs and can walk. These sources of knowledge can be represented in different formats, including algebraic equations, logic rules, and knowledge graphs. Moreover, they can be incorporated into the training process in multiple ways, such as augmenting the training data, modifying the network architecture, adding regularization terms, or altering the optimization process.

For instance, prior approaches have integrated algebraic equations into the training pipeline as an additional loss term [146, 147, 148]. Similarly, logic rules, such as $\{\forall x \in Animals : Fly(x) \wedge LayEggs(x) \rightarrow Bird(x)\}$, have also been incorporated by introducing a semantic loss term into the learning algorithm [149, 150, 151, 152]. Other approaches encode world knowledge into graphs that capture relationships between visual entities, integrating this information into the learning process. For example, Liang et al. [153] proposed inserting a graph reasoning layer into the NN to propagate the information contained in the knowledge graph through subsequent layers. Choi et al. [154] and Zhou et al. [155] employed attention mechanisms on the knowledge graphs to enhance the NN feature representations. However, none of these approaches are designed for the autonomous driving domain, lacking the necessary abstractions to evaluate and enforce the spatio-temporal constraints essential for safe vehicle operation. In Section 4.2, we explain how SD4AS computes safe driving property losses, and how it blends them with the NN main loss by using Lagrangian dual optimization [156].

Chapter 3

Bridging Sensor Inputs and Safe Driving Properties

This chapter addresses a problem in Autonomous Driving System (ADS) safety validation: bridging the semantic gap between low-level, high-dimensional sensor data - that captures the world state - and high-level, human-understandable safe driving properties. We illustrate this concept in Figure 3.1. Defining and evaluating driving properties directly from raw sensor data is difficult because these high-dimensional, low-level world representations inherently lack the necessary semantics required to check complex driving behaviors. At the same time, safe driving rules are expressed using informal natural language, lacking the formalism required for automated evaluation over sensor data. For example, a safe driving rule like "an ADS must not steer toward a concrete barrier" is difficult because it requires not only abstracting low-level sensor data like images and point clouds into semantic concepts like concrete barrier, but also resolving what is the precise spatial meaning of "steering toward".

To address these dual challenges of abstraction and specification, this chapter presents two interconnected contributions, laying the groundwork for improving property conformance throughout the ADS lifecycle, as explored in subsequent chapters.

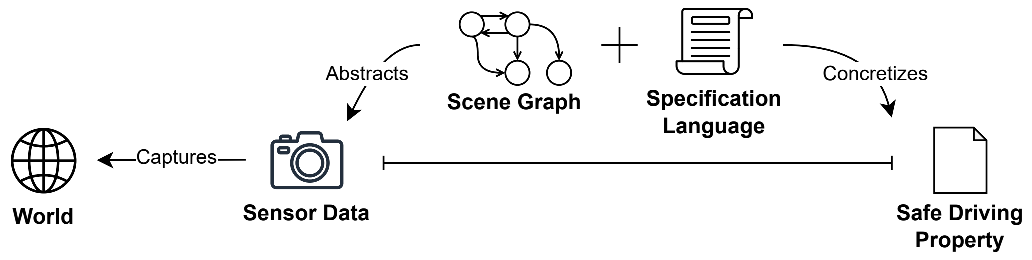


Figure 3.1: Conceptual overview of the gap between sensor data and safe driving properties.

The first contribution focuses on the challenge of abstraction. To bridge the semantic gap, we require an intermediate representation that transforms unstructured, high-dimensional sensor data into high-level semantic entities. The representation must model not just the objects themselves, but also their attributes and the spatial and semantic relationships between them, as these are needed for defining safe driving rules. To meet these requirements, we selected and adapted Scene Graphs (SGs) because they can model a driving scene as a structured graph where nodes encode objects and their attributes, while edges define the relationships between them. This abstraction provides the necessary vocabulary to define and evaluate safe driving properties by analyzing the graph structure.

Building upon this abstraction, the second contribution introduces a specification language to concretize safe driving properties. While SGs provide a semantic representation of the driving scene, specifying safety rules requires a language that can retrieve relevant information by querying the graph. To address this, we designed a Domain-Specific Language (DSL) capable of evaluating semantic propositions against the relational structure of a SG. As we will see later, this DSL can be combined with Linear Temporal Logic over Finite Traces (LTL_f) to specify more complex driving rules that require reasoning about temporal aspects. This combination allows for the precise specification of properties ranging from "never steer left if a barrier is detected on the left" to "once the ego vehicle detects a stop sign controlling its lane, it must stop before passing it". While this chapter focuses on the expressiveness of our approach to specify properties, later ones discuss how we leverage them during development and deployment.

Together, these contributions bridge the gap between sensor data and safe driving properties by first using SGs to abstract the sensor data, and second, providing a language to define properties over these abstractions. They form the basis for the safety validation and enforcement techniques detailed in later chapters. The following sections detail the design of the SG abstraction and the DSL for property specification.

3.1 Scene Graphs

This section details the SG abstraction, the first core contribution of this chapter. We begin by establishing the requirements for an abstraction capable of supporting safe driving properties and justifying why SGs are the best choice among comparable alternatives. Building on this selection, we provide a formal definition of SGs. Finally, to utilize SGs effectively within the SD4AS framework—specifically to enable validation in a controlled environment—we require a tool to obtain reliable and parameterizable, ground-truth scene graph representations. To meet this need, we introduce CarlaSGG, a custom tool we developed to produce ground-truth SGs from the CARLA simulator. This tool provides an accurate source of data, offering many degrees of freedom for customization, and enabling the development of downstream tasks by isolating them from the noise and errors inherent in real-world perception systems.

3.1.1 Selecting the Right Abstraction

To bridge the semantic gap, we must first select an appropriate intermediate abstraction. Safe driving properties cannot be specified directly over raw sensor data because a property like "do not steer toward a concrete barrier" requires recognizing high-level concepts—such as barrier and its spatial location—which are not directly present in high-dimensional sensor outputs that inherently lack such semantics. Obtaining these high-level concepts requires transforming sensor data into a higher-level representation. To identify the most appropriate choice, we analyze several potential abstractions in Table 3.1 based on four criteria needed for the AV domain.

First, "Property Definition & Evaluation" assesses whether the abstraction provides a structured representation to driving specifications. A structured representation ensures that data is organized consistently, using a single vocabulary for object names and their relations. This structure enables

Abstraction	**Property Definition & Evaluation**	**Spatial Information**	**Object Attributes**	**Relationship between Objects**
Natural Language	✗	●	●	✗
2D Named BB	✓	●	✗	✗
3D Named BB	✓	✓	✗	✗
Scene Graphs	✓	✓	✓	✓

Table 3.1: Comparison of different scene abstractions based on four criteria required for specifying and evaluating safe driving properties.

automating property evaluation because it provides a fixed schema that allows rules to target precise and predictable variables—such as a standardized "barrier" class—rather than requiring the system to interpret varying or ambiguous descriptions. Structured abstractions, such as named Bounding Boxes (BB), satisfy this criterion by providing concrete concepts—like object classes—that can be mapped to the variables typically used for specifying driving properties. In contrast, unstructured abstractions, like natural language descriptions which can be generated by a VLM, are ambiguous and may not reliably contain the specific information required for automatically evaluating properties.

Second, "Spatial Information" considers whether the abstraction captures sufficient details about all relevant property dimensions. Many safety properties require reasoning about euclidean distance between objects, which is not possible with 2D named BBs that lack depth information (the z coordinate). Similarly, natural language descriptions are unsuitable because they often rely on qualitative terms like "close" or "far". These terms are ambiguous because their interpretation is subjective—varying between individuals or contexts—and thus they lack the actual measurement required for automated safety checks.

Third, "Object Attributes" assesses the ability to capture different characteristics of the entities which is beyond simple object classification. Safe operation requires knowing not just that an object exists, but its specific attributes—for example, distinguishing whether a traffic light is Red or Green, or if a vehicle has its blinkers on. Standard detection outputs (2D or 3D BBs) typically provide only the object category, location, and sometimes dimension estimates, missing these other attributes. While natural language descriptions can capture these details, they often do so inconsistently and in an unstructured format. For instance, a generated text might mention a "traffic light" without specifying its color. Even when the description includes this information, extracting it for evaluation requires custom parsing rather than a direct lookup. This manual processing remains necessary unless the model produces the output in a strict format, such as JSON.

Finally, "Relationship between Objects" assesses the capability to capture the semantic context by explicitly linking entities. Safe operation depends not just on identifying objects, but on understanding their interactions—for example, determining whether a specific traffic light controls the ego vehicle's lane, or if another vehicle is directly $inFrontOf$ the ego vehicle. Standard object detectors (2D and 3D) fail this criterion as they generate isolated lists of objects without encoding how they relate to one another. Similarly, while natural language can describe relationships, it does not provide explicit, directed links that can be reliably queried to validate these interactions.

Based on these criteria, we analyze the limitations of alternatives abstractions. Natural Language, while high-level, is unstructured and often lacks precise spatial data, object attributes, and explicit relationships between objects, making it unsuitable for automated evaluation. 2D Detection provides structured data for evaluation but offers only partial spatial information, as it lacks depth, no object attributes, and not relationships between the objects. 3D Detection solves the spatial challenge by providing full 3D coordinates, but it fails to capture object attributes and relationships between objects. SGs, however, satisfy all four criteria. They provide a structured representation where nodes encapsulate both the full 3D spatial information and the specific state attributes of every object. Furthermore, they explicitly encode the semantic and spatial interactions between these entities through directed edges. This makes them a suitable abstraction for defining and evaluating safe driving properties.

3.1.2 Scene Graph Definition

A Scene Graph (SG) is a directed graph that encodes the semantic relationships between objects in a scene. Formally, $SG = (V, E : V \mapsto V, kind : V \mapsto K, rel : E \mapsto R, att : V \cup E \mapsto M)$ where V is the set of nodes representing the entities in a scene, E is a set of directed edges, *kind* is a function to access the entity type of a node, *rel* is a function to get the relation of an edge, and *att* is a function to retrieve the attribute values of a node or edge.

In this work, we utilize different encodings to represent this formal definition. While the primary encoding utilizes a directed graph, we also employ triplets, to represent the basic constituents of the graph. In this format, an SG is defined as a set of triplets in the form of <subject, relation, object> or <object, "is", attribute>, where subject and object are nodes in the graph connected by a directed edge labeled with relation. Likewise, objects can have different attributes linked through the "is" relationship. The triplet representation is particularly useful for tasks that require sequential formats, such as natural language, as it facilitates mapping the graph structure to text as we explain in Section 5.2.

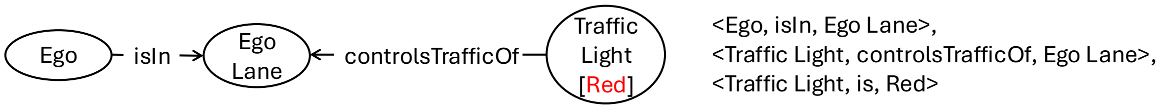


Figure 3.2: Example of SG as a directed graph and a set of triplets

In this dissertation, we focus our attention on SGs tailored for the AV domain. These SGs always contain a distinguished *Ego* node ($Ego \in V$), representing the autonomous vehicle from whose perspective the SG is generated, and an *Ego_Lane* node ($Ego\_Lane \in V$), representing the lane ego is in. Figure 3.2 provides a simple SG example in the AV domain, illustrating both definitions. The graph on the left depicts an ego node that is in an ego lane node, whose traffic is controlled by a traffic light node that is red. Similarly, the triplets on the right describe the same scene by detailing two relationships between nodes and one node attribute.

Finally, we define the process to create an SG. A Scene Graph Generator (SGG) is a mapping from a set of sensor inputs, T, to an SG representation, formally denoted as $sgg : T \mapsto SG$. This definition establishes the theoretical basis for the practical implementation presented in the following subsection, where we detail CarlaSGG —our custom SGG designed to produce ground-truth graphs from the CARLA simulator.

3.1.3 CARLA Scene Graph Generator

Validating the downstream tasks within the SD4AS framework requires a reliable and accurate source of SG data. Using a simulation environment is advantageous for this purpose, as it provides access to ground-truth information, thereby decoupling the SG generation process from the complexities and errors inherent in real-world perception. We selected the CARLA simulator [157] to serve as the simulation platform for validating SD4AS downstream tasks and generating ground-truth SGs. CARLA is an open-source platform widely used in the development and testing of autonomous driving systems. The platform provides open digital assets, flexible specification of sensor suites, and full control over all actors. Furthermore, CARLA includes multiple preconstructed maps designed to mimic various driving environments, which allows for data collection across diverse scenarios. These capabilities support the validation required for ADS research [158].

To generate SGs within this environment, we developed CarlaSGG[1], a tool that extracts ground-truth SGs directly from the CARLA simulator. By querying the internal simulation engine directly, the tool bypasses perception errors to provide an exact representation of the environment. Operationally, the plugin is configured to focus on a predefined set of entity types—including roads, signs, vehicles, and other relevant actors—allowing it to selectively capture the relevant portion of the simulation state after every time step. It then synthesizes a SG according to a set of parameterizable constraints, utilizing configurable abstraction layers to tailor the content to suit the requirements of a property specification—such as capturing traffic light state if a rule requires identifying red lights. This process converts the dynamic simulation state into a structured graph suitable for formal analysis.

Figure 3.3 illustrates the flow of our tool to generate SGs from CARLA. Throughout this section we will use *client* to refer to the program utilizing the CARLA plugin, CarlaSGG, along with CARLA to collect data. The *client* instantiates CARLA with the relevant environment and sensor configurations, and controls the flow of the simulation. For example, the *client* manages the simulation time-steps and sends control commands (e.g., throttle and steering) to navigate the ego vehicle through the environment. At any point, the *client* can utilize CarlaSGG to generate the SG representing a scene with entities, attributes, and their relationships at that instant in time. The plugin operates in three phases; we describe each phase below.

[1]We make our code available at: https://github.com/less-lab-uva/carla_scene_graphs

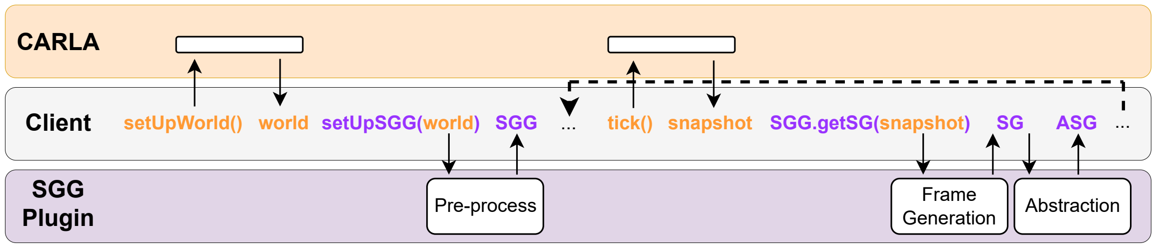


Figure 3.3: CarlaSGG workflow for generating SGs within the CARLA simulator.

3.1.3.1 Pre-processing

For each simulation, CARLA first loads the static background environment, including the roadway, buildings, signage, and scenery. These environments can be vast, with the largest maps approaching 10,000 hectares. For efficiency, CarlaSGG first performs a pre-processing pass over the environment, computing an initial SG containing the road structure and all static entities. The road structure is encoded as a high-definition map, with its resolution selected by the *client*; each node represents a location in the roadbed with edges describing the semantics of traffic flow at that location. For example, an edge connecting two consecutive nodes along the same path might be labeled as "lane_follow", while an edge connecting nodes in adjacent lanes might be labeled "lane_change" to indicate a permitted maneuver. The remaining static entities are extracted and their spatial and semantic relationships pre-computed to form the initial SG. This cached static background SG will allow the tool to efficiently generate a local SG at each frame that is centered on the ego vehicle, containing only the information about its immediate surroundings rather than the entire map.

3.1.3.2 SG Generation per Frame

At each frame, or at the desired frequency established by the *client*- every n frames - CarlaSGG can be invoked to generate a local SG capturing the semantics of the current environment around the ego vehicle ("world snapshot" in CARLA terms). To construct the SG from the perspective of the distinguished *Ego* node (as defined in Section 3.1.2), the cached static background SG is first filtered to produce a subgraph, retaining only the static elements within a user-specified region (e.g., a rectangular BB or a radius), chosen by the *client*, centered on the ego vehicle. This subgraph is then

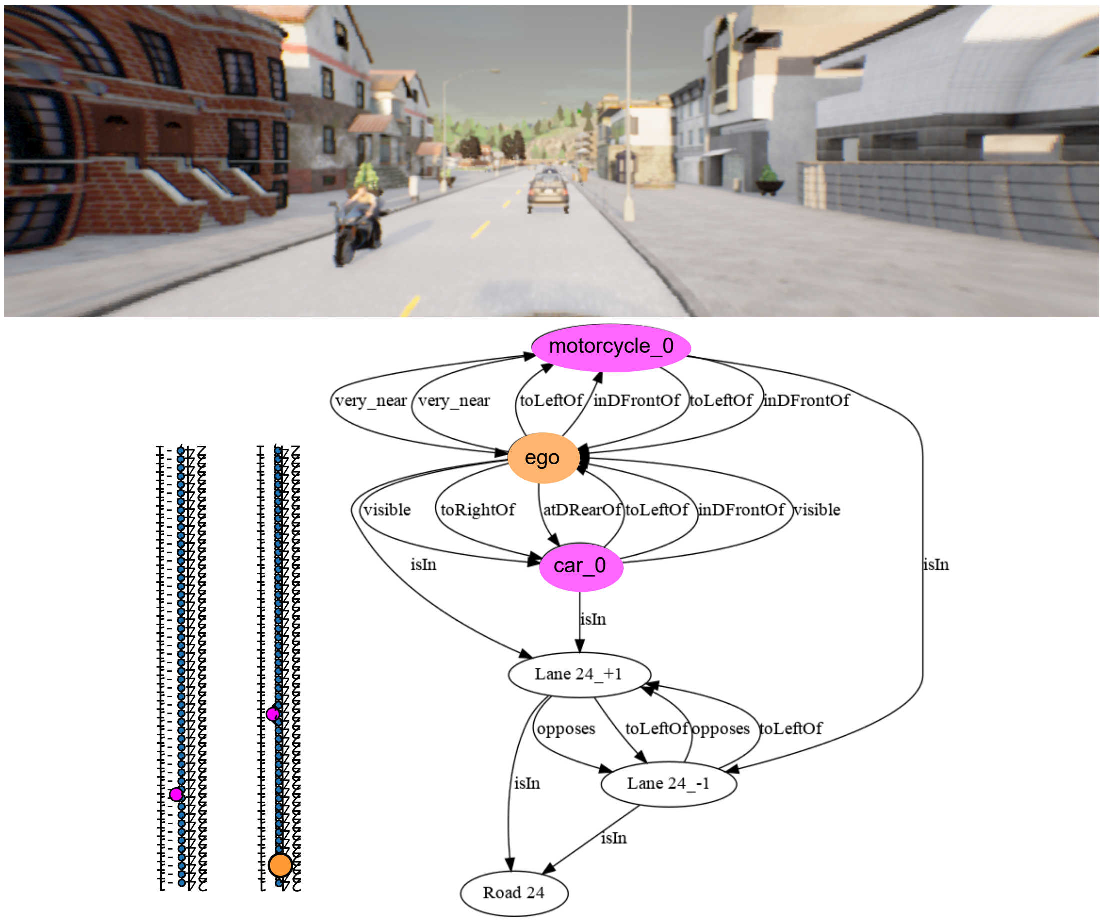


Figure 3.4: A simulator image (top), its SG (left), and an abstracted SG (right).

enriched by adding all dynamic entities—such as other vehicles (cars, trucks, buses), pedestrians, and cyclists—that are currently within that same region, annotating them with their attributes like position and velocity. The resultant SG contains all static and dynamic entities relevant to the ego vehicle's current situation.

3.1.3.3 Abstraction

The SG generated in the previous steps contains many fine-grained details that may not be relevant for the *client*'s particular use case. As illustrated in Figure 3.4, the SG on the left contains dozens of nodes representing the road structure (blue), along with two nodes representing the other entities in the roadway (magenta), and one node representing the ego vehicle (orange). While this may be suitable for tasks involving low-level motion-planning, this information may be superfluous for higher-order tasks, such as computing scenario coverage, re-training DNNs for property compliance, or monitoring and enforcement of safe driving properties. To this end, CarlaSGG provides the functionality to *abstract* the SG to create an Abstracted Scene Graph (ASG) that lifts the semantic information contained in the original SG to a higher-level semantics. As shown in Figure 3.4, in the ASG on the right, CarlaSGG has abstracted away the individual nodes representing the roadbed and replaced them with nodes representing the lanes and roads while preserving the semantic information that these lanes oppose each other in the flow of traffic.

After establishing the nodes based on the chosen abstraction level, the tool computes the pairwise spatial and semantic relationships to form the edges of the ASG. This process iterates over the relevant entity pairs to determine relationships such as relative position (e.g., inFrontOf, toRightOf), distance (e.g., within7m, between7_10m), lane relationships (e.g., isIn), and signal-to-lane relationships (e.g., controlsTrafficOf). Table 3.2 lists the built-in abstractions provided by CarlaSGG, which include an abstraction to match the level of semantic information present in RoadScene2Vec [70] (shown in Figure 3.4), as well as higher-order abstractions to remove the lane and/or relationship information from the graph. Additionally, the abstraction functions are extensible, and client applications can design additional abstractions as needed.

Abstraction	**Description**
Entities (E)	Includes only entities (e.g., vehicles, pedestrians) in the scene, similar to a semantic segmentation output.
Entities + Lanes (EL)	Includes entities (E) and adds information about which lane each entity occupies.
Entities + Relations (ER)	Includes entities (E) and adds inter-entity relationships (e.g., 'inFrontOf', 'within7m') based on the RoadScene2Vec configuration.
Entities + Lanes + Relations (ELR)	Combines EL and ER: includes entities, their lane occupation information, and inter-entity relationships.
Entities + Road Structure (ERS)	Similar to EL, but models the road structure with high-fidelity, representing lanes as multiple fine-grained road segments.

Table 3.2: SG Abstractions in CarlaSGG.

3.1.3.4 Validation and Adoption

We have extensively validated CarlaSGG by using it to collect millions of SGs, representing days of simulated driving captured across most of the available CARLA towns. Its utility is further evidenced by its role in our research. It has been a key component in five accepted publications [30, 28, 159, 29, 31] and one work currently under submission. These works utilize the generated SGs for diverse applications, such as dataset coverage analysis, runtime monitoring of safe driving properties, and training DNNs for property conformance. The tool remains actively maintained and available as an open-source project to support future research.

3.1.3.5 Limitations

The primary limitation of CarlaSGG is its reliance on a simulation environment. By design, the tool operates on privileged, ground-truth information directly from the simulator's state, thereby bypassing critical real-world perception challenges such as sensor noise, occlusions, and the sim-to-real domain gap. Also, because the tool extracts entities based on their geometric presence within a specified radius rather than line-of-sight, it currently lacks an occlusion filtering mechanism. Consequently, the generated SGs may include entities that would be invisible to physical sensors—such as a pedestrian completely obscured by a truck—resulting in a representation that contains imperceptible objects. This positions the tool for offline development and validation within a controlled setting, rather than for direct deployment on a physical vehicle. Consequently, the fidelity and diversity of the generated SGs are inherently bounded by the scenarios and assets available within CARLA, which in turn limits the generalizability of any downstream module validated with this SGG to conditions reproducible within the simulation. Further, although CARLA is widely used in the AV testing and development space, the simulation-reality gap [160] remains a limitation in generalizing results to real-world performance.

A second limitation is the difficulty in generating abstractions for complex road topology, particularly intersections. Intersections are inherently complex, with multiple lanes merging, crossing, and diverging. This can cause the resulting graph abstractions to become overly-detailed or ambiguous.

For example, a high-fidelity abstraction might represent a single four-way intersection with dozens of discrete road segment nodes. This level of detail makes queries like "is there a vehicle in the same lane as ego vehicle" challenging to evaluate. As the ego vehicle traverses the intersection, it may logically occupy several of these lane segments, meaning any car on any of those segments could be considered in the ego's lane. This complexity can make it difficult for downstream tasks to query and reason about intersection-level semantics, such as identifying objects in specific lanes or determining if the ego vehicle is in an opposing lane.

3.1.4 Summary

This section defined the SG as a structured representation for driving environments and introduced CarlaSGG, a tool to generate these graphs from simulation. Using a SGG from ground-truth data allows the development of high-level downstream tasks to be separated from the challenges of real-world perception. With this method for scene representation established, the following section introduces the formal language used to specify and evaluate safe driving properties over these SGs.

3.2 Safe Driving Property Specification

Building upon the structured scene representation introduced in the previous section, this section formalizes how safe driving properties can be precisely defined and their semantic predicates evaluated over SGs. To that end, we introduce Scene Graph Language (SGL)—a domain-specific language based on formal logic that interacts with SG to evaluate semantic predicates, and can be combined with Linear Temporal Logic over Finite Traces (LTL_f) to capture temporal aspects. SGL enables the specification of interpretable and temporal safe driving properties directly over the entities, attributes, and relationships captured in the SG. Through this formulation, driving laws can be encoded as formal specifications, forming the basis for reasoning about safety compliance in autonomous driving scenarios.

3.2.1 Scene Graph Language

Scene Graph Language (SGL), is a Relational First Order Language (RFOL) with a set of SG querying functions, to facilitate the specification of driving properties. An expression in SGL is a RFOL formula in which the propositions are symbolic graph queries. In an SGL expression, AP is defined by a graph query, $AP : SG \mapsto Boolean$, built up out of Boolean expressions (B) in turn built up out of expressions defining sets of SG vertices (S):

$$\begin{aligned}
AP \;&::= (\neg B) \mid (B_1 \wedge B_2) \mid (B_1 \vee B_2) \mid (B_1 \implies B_2) \mid (B_1 \oplus B_2)\\
B \;&::= (\|S\| > N) \mid (\|S\| < N) \mid (\|S\| \geq N) \mid (\|S\| \leq N) \mid (\|S\| = N) \mid \mathit{false} \mid \mathit{true}\\
S \;&::= sg.V \mid (S_1 \cup S_2) \mid (S_1 \cap S_2) \mid (S_1 \setminus S_2) \mid (S_1 \triangle S_2) \mid \mathit{relSet}(S, r) \mid\\
&\quad\; \mathit{relSetR}(S, r) \mid \mathit{filterByAttr}(S, m, f)
\end{aligned}$$

Where $\neg, \wedge, \vee, \implies$, and $\oplus$ are the logic not, and, or, implication, and exclusive or operators respectively; $\|\cdot\|, \cup, \cap, \setminus$, and $\triangle$ are the set size, union, intersection, difference, and symmetric difference operators respectively; $>, <, \geq, \leq$, and $=$ are the greater than, less than, greater than or equal, less than or equal, and equality test operators; $N \in \mathbb{N}$. In the semantics, S is a set of vertices in the SG and $sg.V$ is the set of all vertices in the graph ($S \subseteq sg.V$). We can use this syntax to define a standard expression that is common to many SGL expressions: the special set $Ego = \mathit{filterByAttr}(sg.V, name, \text{“}ego\text{”})$, which is the set containing the lone vertex for referring to the ego vehicle in the SG.

To enable the definition of APs over a SG, SGL provides a set of graph querying primitives that operate directly on the entities and relations of the SG. These core functions—**relSet**, **relSetR**, and **filterByAttr**—allow users to traverse relational edges, access reverse relations, and filter vertices based on semantic or numeric attributes. Together, these core functions will be used to construct APs that capture spatial, relational, and attribute-based conditions within the driving scene, and are defined as follows:

The **relSet** computes the join of a set of vertices and a relation.

$$relSet : (V_1 \subseteq V, r \in R) \mapsto V_2 \subseteq V$$
$$V_2 = \{v_2 : v_1 \in V_1 \wedge (v_1, v_2) \in E \wedge rel(v_1, v_2) = r\}$$

For example, the set of lanes controlled by a stop sign is $relSet(stopSigns, \text{controlsTrafficOf})$. SGL also supports **relSetR**, which is the join of the transpose of the given relation, e.g., the set of stop signs controlling a given lane is $relSetR(lane, \text{controlsTrafficOf})$.

The **filterByAttr** selects a subset of vertices, V_1, whose attribute, m, satisfies a given predicate,f.

$$filterByAttr : (V_1 \subseteq V, m \in M, f : T \mapsto bool) \mapsto V_2 \subseteq V$$
$$V_2 = \{v : v \in V_1 \wedge type(att(v)[m]) = T \wedge f(att(v)[m])\}$$

For example, $filterByAttr(trafficLights, \text{lightState}, \lambda x : x = \text{Red})$ yields the red traffic lights. More examples of APs defined using SGL can be seen in Table 3.6.

We can use these SGL querying primitives to identify specific entities and relations in the environment that require a response from the ADS, and define a set of safe driving properties, P. A safe driving property is an implication of the form $\phi_{\mathcal{X}} \implies \phi_{\mathcal{Y}} \in P$ where the precondition, $\phi_{\mathcal{X}}$, is specified using SGL over the SG extracted from the raw sensor input, and the postcondition, $\phi_{\mathcal{Y}}$, is defined as a conjunction of interval constraints in the ego vehicle's output space. For instance, *if there is a red light, ego vehicle should stop*, could be encoded as $hasStop \implies ego.acceleration < 0$, where $\phi_{\mathcal{X}} = hasStop$ and $\phi_{\mathcal{Y}} = ego.acceleration < 0$. More properties preconditions are studied in Section 4.1, Section 4.2, Section 5.2 and Section 5.3.

3.2.2 SGL + LTL$_f$

The properties with the implication form, defined using pure SGL, can only evaluate conditions at a single timestep, restricting the specification to static rules based on the current observation. However, many driving scenarios require analyzing how the environment changes over a sequence of steps. To capture these temporal aspects and enable a precise characterization of the AV's actions and responses in dynamic environments, we integrate SGL with LTL_f. In this integration, the SGL expressions function as APs. Building from the definition of an LTL_f formula [97], an expression ϕ in SGL is:

$$\phi ::= AP \mid (\neg\phi) \mid (\phi_1 \wedge \phi_2) \mid (\phi_1 \vee \phi_2) \mid (\phi_1 \implies \phi_2) \mid (\mathcal{G}\phi) \mid (\mathcal{F}\phi) \mid (\mathcal{X}\phi) \mid (\phi_1 \mathcal{U} \phi_2) \mid \$[N][AP]$$

Where $\mathcal{G}, \mathcal{F}, \mathcal{X}$, and $\mathcal{U}$ are the standard LTL_f operators discussed in Section 2.2.2. Additionally, SGL defines a discrete metric operator, \$[N][AP], to ease the LTL specification over repeated APs by unrolling the AP N times using the $\mathcal{X}$ operator. The expressiveness of SGL and LTL_f properties is studied next in Section 3.2.3 and their evaluation is studied in Section 5.1.

3.2.3 Expressiveness

To demonstrate the capability of SGL with LTL_f for encoding safe driving properties relevant to AV systems, in a recent paper [29], we analyzed 207 numbered sections of the Virginia Driving Code [161] and showed that 76.0% of the ones applicable to AVs can be encoded. As illustrated in Figure 3.5 and Table 3.3[2], 114 (55.1%) of these sections from the driving code are applicable to typical autonomous systems. The remaining 93 (44.9%) sections handle bureaucratic administration or do not apply to typical autonomous systems, e.g. § 46.2-812 states *"No person shall drive ... for more than thirteen hours in any period of twenty-four hours"*. Of these 114 applicable sections, 87 (42.0%) can be fully expressed by SGL + LTL_f, while an additional 8 (3.9%) are partially expressible, as the numbered sections contain multiple clauses and some of them are expressible. The remaining 19 properties (9.2%) are currently inexpressible. Four of these are inexpressible due to the imprecise language of the specification, such as § 46.2-864 which prohibits "reckless driving" defined as *"[operating] any motor vehicle at a speed or in a manner so as to endanger the life, limb, or property of any person"* [161]. The other 15 properties require tracking specific entities over time, which is a limitation later discussed in Section 3.2.4.

[2]Adapted from "Scene Flow Specifications: Encoding and Monitoring Rich Temporal Safety Properties of Autonomous Systems" [29]

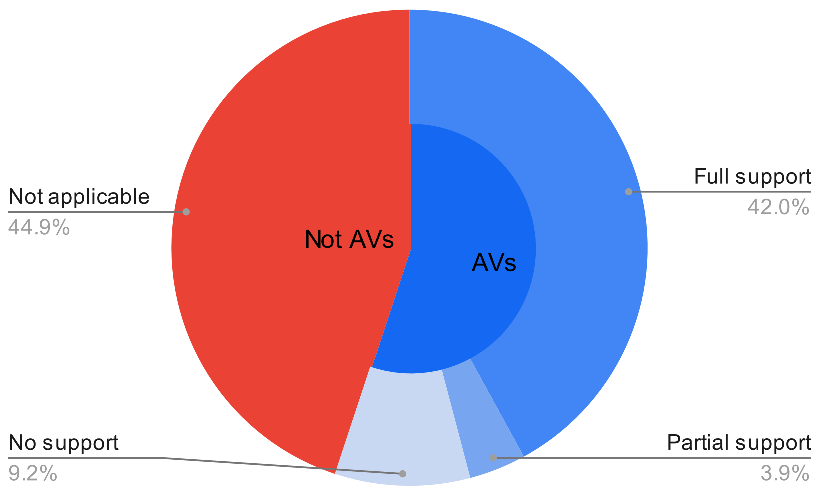


Figure 3.5: Piechart showing the percentages of the VA driving rules that can be expressible, partially expressible and not expressible using SGL + LTL$_f$.

For AV?	Express in SGL + LTL$_f$?	#	Sections within § 46.2
✗	—	93	800, 800.2, 800.3, 801, 808, 808.2, 808.3, 809, 809.1, 810, 810.1, 811, 812, 813, 815, 816.1, 817, 818.2, 819, 819.1, 819.2, 819.3, 819.3:1, 819.4, 819.5, 819.6, 819.7, 819.8, 819.9, 819.10, 830.1, 830.2, 831, 832, 833.01, 840, 844, 853, 855, 860, 861, 866, 867, 868, 869, 872, 874.1, 876, 878, 878.3, 879, 880, 882, 882.1, 883, 891, 895, 896, 897, 898, 899, 900, 901, 902.1, 904.1, 906.1, 908, 911, 913, 915, 916, 916.2, 917.1, 917.2, 918, 919, 919.1, 920.1, 920.2, 921.1, 931, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 944.1
✓	✓	87	800.1, 802, 803.1, 805, 806, 807, 808.1, 814, 818, 818.1, 825, 827, 828, 828.1, 828.2, 830, 834, 835, 836, 838, 841, 845, 848, 849, 850, 851, 857, 859, 861.1, 862, 870, 871, 873, 873.1, 873.2, 874, 875, 877, 878.1, 878.2, 878.2:1, 881, 884, 885, 886, 887, 888, 889, 890, 892, 893, 894, 902, 903, 904, 906, 908.1, 908.1:1, 908.2, 908.3, 909, 910, 911.1, 912, 914, 915.1, 915.2, 916.1, 916.3, 917, 922, 923, 925, 926, 927, 928, 929, 930, 932, 932.1, 933, 822, 824, 826, 863, 846, 847
	◐	8	803, 804, 821, 833, 905, 907, 920, 924
	✗	19	816, 820, 823, 829, 833.1, 837, 839, 842, 842.1, 843, 854, 856, 858, 865, 921, 852, 864, 865.1, 868.1

Table 3.3: Rules from the Virginia driving code [161] supported by SGL + LTL$_f$
✗= No Support, ◐= Partial Support, ✓= Full Support.

From the set of expressible rules, we selected 9 representative properties derived to specify using SGL + LTL$_f$, as detailed in Table 3.4. These laws were chosen because they are within the scope of current AV systems and exhibit diversity in both the temporal aspects required for property compliance analysis and the richness of the SG structure needed to evaluate the atomic propositions.

ϕ	VA Code	English Summary of Property	LTLf Formula over SG propositions
ϕ_1	§ 46.2-804	Ego vehicle cannot be in the opposing lane	$\mathcal{G}(\neg isOppLane)$
ϕ_2	§ 46.2-802	Ego vehicle cannot be out of the road.	$\mathcal{G}(\neg isOffRoad)$
ϕ_3	§ 46.2-802	If ego vehicle is in the rightmost lane, then ego vehicle should not steer to the right.	$\mathcal{G}(isInRightLane \land \neg isJunction \rightarrow isNotSteerRight)$
ϕ_4	§ 46.2-816	Ego vehicle should not be behind another entity in the same lane whithin 4 meters while travelling at a speed $> S$.	$\mathcal{G}(isNearColl \rightarrow \neg isFasterThanS)$
ϕ_5	§ 46.2-816	If ego vehicle is between 4 and 7 meters of the closest vehicle in the same lane and then comes within 4 meters of a vehicle in the same lane, throttle must not be positive.	$\mathcal{G}((isSuperNear \land \neg isNearColl) \land \mathcal{X}(isNearColl) \rightarrow \mathcal{X}(isNoThrottle))$
ϕ_6	§ 46.2-888	If the ego vehicle is moving and there is no entity in the same lane as the ego vehicle within 7 meters, and there is no red traffic light or stop sign controlling the ego vehicle's lane, then the ego vehicle should not stop.	$\mathcal{G}(\neg isStopped \land \neg(isSuperNear \lor isNearColl) \land \neg hasRed \land \neg hasStop \land \mathcal{X}(\neg(isSuperNear \lor isNearColl) \land \neg hasRed \land \neg hasStop) \rightarrow \mathcal{X}(\neg isStopped))$
ϕ_7	§ 46.2-804	If ego vehicle is not in a junction, then ego vehicle cannot be in more than one lane for more than T seconds (N samples).	$\neg\mathcal{F}\$[N][isMultipleLanes \land \neg isJunction]$
ϕ_8	§ 46.2-833	Ego vehicle must exit junctions within T seconds (N samples).	$\neg\mathcal{F}\$[N][isOnlyJunction]$
ϕ_9	§ 46.2-821	Once the ego vehicle detects a new stop signal controlling its lane, it must stop before passing the stop signal.	$\mathcal{G}((\neg hasStop \land \mathcal{X}(hasStop)) \rightarrow (\mathcal{X}(hasStop\ \mathcal{U}\ (isStopped \lor \mathcal{G}(hasStop)))))$

Table 3.4: Properties implemented using SGL + LTL$_f$.

Table 3.4 shows these 9 properties, with their relevant statute, a short English summary, and their encoding using the APs over the SG composed through the LTL$_f$ formula. The specific APs and intermediate SGL variables used in these encodings are detailed in Table 3.5 and Table 3.6, respectively. We note that precisely encoding the semantics of the law is challenging. To illustrate this, consider the stop sign property, ϕ_9. The APs are evaluated over the LTL$_f$ formula to track if *isStopped* is true at least once between *hasStop* becoming true and later becoming false, indicating that ego stopped while being controlled by the stop sign. This is a necessary but insufficient specification to meet the criteria under the law; notably, this does not check that the vehicle stopped *at the stop line* rather than before, nor does it enforce separate stops for successive stop signs along the same lane.

We note that while some parameters can be expressed in SGL like speed, S, in ϕ_4 or time, T, in $\phi_7 - \phi_8$, others are reliant on the parameterization of the underlying SGG. In ϕ_4, ϕ_5, and ϕ_6, we use 4 and 7 meters as the distance thresholds because these correspond to the 'near collision' and 'super near' relationship used by prior AV SGGs [63]. Further, the underlying laws do not provide

Atomic Prop.	SGL expression
isJunction	$\lvert egoJunctions \rvert > 0$
isOppLane	$\lvert oppLanes \rvert > 0$
isOffRoad	$\lvert offRoad \rvert > 0$
isInRightLane	$\lvert rightLanes \rvert = 0$
isNotSteerRight	$\lvert steerRight \rvert = 0$
isNearColl	$\lvert nearColl \rvert > 0$
isFasterThanS	$\lvert egoFasterS \rvert = 1$
isSuperNear	$\lvert superNear \rvert > 0$
isNoThrottle	$\lvert noThrottle \rvert = 1$
isMultipleLanes	$\lvert egoLanes \rvert > 1$
hasRed	$\lvert redLightLanes \cap egoLanes \rvert > 0$
hasStop	$\lvert stopSignLanes \cap egoLanes \rvert > 0$
isStopped	$\lvert egoStopped \rvert = 1$
isOnlyJunction	$\lvert egoRoads \setminus juncRoads \rvert = 0$

Table 3.5: Atomic Propositions

Name	SGL expression
egoLanes	$relSet(Ego, \text{isIn})$
egoRoads	$relSet(egoLanes, \text{isIn})$
egoJunctions	$relSet(egoRoads, \text{isIn})$
oppLanes	$relSet(egoLanes, \text{opposes})$
offRoad	$filterByAttr(egoLanes, \text{kind}, \lambda x : x = \text{offRoad})$
rightLanes	$relSet(egoLanes, \text{toRightOf})$
steerRight	$filterByAttr(Ego, \text{steer}, \lambda x : x > 0)$
inEgoLane	$relSetR(egoLanes, \text{isIn}) \setminus \{Ego\}$
nearColl	$relSet(inEgoLane, \text{near\_coll})$
superNear	$relSet(inEgoLane, \text{super\_near})$
egoFasterS	$filterByAttr(Ego, \text{speed}, \lambda x : x > S)$
noThrottle	$filterByAttr(Ego, \text{throttle}, \lambda x : x < \epsilon)$
trafficLights	$filterByAttr(sg.V, \text{kind}, \lambda x : x = \text{trafficLight})$
redLights	$filterByAttr(trafficLights, \text{lightState}, \lambda x : x = \text{Red})$
redLightLanes	$relSet(redLights, \text{controlsTrafficOf})$
stopSigns	$filterByAttr(sg.V, \text{kind}, \lambda x : x = \text{stopSign})$
stopSignLanes	$relSet(stopSigns, \text{controlsTrafficOf})$
egoStopped	$filterByAttr(Ego, \text{speed}, \lambda x : x < \epsilon)$
juncRoads	$relSetR(egoJunctions, isIn)$

Table 3.6: Intermediate variables used in Atomic Propositions shown in Table 3.5

concrete values, e.g. the law from ϕ_5 says "[...] a motor vehicle shall not follow [a vehicle] more closely than is reasonable and prudent [...]" [161].

3.2.4 Limitations

The difficulty of translating ambiguous, qualitative driving laws into precise, quantitative specifications challenges the correctness of the property specifications. Driving manuals often use imprecise terms, such as requiring a vehicle to follow another at a "reasonable and prudent" distance. Concretizing these rules into formal atomic propositions requires developers to select specific thresholds (e.g., 7 meters), which may not capture the full context-dependent nuance of the original rule. This can also lead to specifications that are necessary but insufficient to fully comply with the law. For example, the stop sign property ϕ_9 verifies that the vehicle stops but does not enforce the spatial requirement that the stop occur precisely at the stop line.

A second limitation, previously noted in Section 3.2.3 regarding the 15 inexpressible properties, concerns tracking specific entities across temporal states. The conversion of rich graph queries into Boolean atomic propositions for LTL$_f$ results in information loss regarding explicitly which entities satisfied the query. Consider Virginia Code § 46.2-816, which prohibits following another vehicle more closely than is reasonable. An approximation using SGL might define an AP, *isTooClose*, that is true whenever the ego vehicle is within a certain distance of *any* other vehicle. A temporal rule forbidding *isTooClose* over consecutive time steps, $\mathcal{G}(\neg(\mathit{isTooClose} \wedge \mathcal{X}\ \mathit{isTooClose}))$, would incorrectly flag a violation if the ego vehicle was close to one vehicle at time t and a *different* vehicle at time $t + 1$, perhaps due to a lane change. Similar issues arise with right-of-way rules, such as § 46.2-820, which requires yielding to vehicles arriving at an intersection simultaneously from the right. Effectively monitoring this requires tracking *which* specific actor established precedence to ensure the ego vehicle yields to the correct entity over time, rather than just monitoring if *any* vehicle is currently on the right.

3.2.5 Summary

This section introduced SGL a domain-specific language for specifying safe driving properties over the SG abstraction. We validated the expressiveness of SGL by analyzing the Virginia Driving Code, demonstrating that 76% of applicable statutes could be encoded. To illustrate practical utility, we detailed the implementation of nine representative properties requiring diverse temporal and structural reasoning. Finally, we discussed inherent limitations in formalizing ambiguous human laws and the challenges of tracking specific entities across time with Boolean atomic propositions.

3.3 Conclusion

This chapter connected raw sensor data and high-level safe driving properties through two interconnected contributions: the Scene Graph (SG) abstraction and the Scene Graph Language (SGL). These contributions bridge the semantic gap by first abstracting the sensor data into a SG, and then utilizing SGL to formally specify driving properties over this structured representation. These contributions form the basis of the SD4AS framework, which the subsequent chapters will apply to enhance safety throughout the ADS development and deployment."

Chapter 4

Increasing property conformance in ADS development

This chapter addresses the problem of proactively embedding safety into the development lifecycle of ADSs. Resolving this problem requires tackling two challenges: ensuring that training data adequately covers the spectrum of safety-critical scenarios an ADS will encounter, and ensuring that the training process compels the model to internalize safe behaviors within those scenarios. To address these challenges, this chapter details the development-focused portion of the SD4AS framework, which is composed of two contributions, S^{3}C and T4PC.

The first contribution, S^{3}C (**S**patial **S**emantic **S**cene **C**overage), addresses the challenge of data adequacy. Recognizing that simply collecting large volumes of driving data offers no assurance of covering rare but critical "corner cases", S^{3}C provides an automated way to quantify dataset coverage against specified properties. By leveraging SGs to create interpretable, semantic abstractions of raw sensor inputs, S^{3}C enables developers to analyze the diversity of their datasets, identify specific gaps in coverage, and make informed decisions about data collection and augmentation. This ensures the ADS has access to the necessary scenarios to learn from.

Building upon a well-curated dataset, the second contribution, T4PC (**T**raining **f**or **P**roperty

Conformance), tackles the challenge of ensuring the ADS learns the correct behaviors. Exposing a model to relevant data alone is insufficient, as standard training objectives do not explicitly penalize unsafe actions. T4PC addresses this by integrating a custom, property-based loss function directly into the DNN's training pipeline. This approach penalizes deviations from specified safe driving rules and uses Lagrangian optimization to balance the dual objectives of achieving high accuracy on the primary driving task and ensuring strong conformance with safety properties, guiding the model towards safer decision-making.

Together, these two contributions enable the SD4AS framework to enhance ADS safety during development. S^3C ensures the model is trained on a semantically rich and diverse dataset, while T4PC ensures the model learns to conform to safety properties when encountering those scenarios. The following sections will detail the design, and empirical evaluation of each contribution, demonstrating their effectiveness in increasing property conformance in ADS development.

4.1 Scene Coverage for Autonomous Vehicles

Ensuring Autonomous Vehicle (AV) safety and reliability requires testing across a range of driving scenarios, which creates a challenge in quantifying whether the data used for training and testing adequately covers the diverse semantic and spatial configurations of the driving environment. Collecting large volumes of driving data is inefficient and offers no guarantee of covering the rare "corner cases" that often lead to system failures. This limitation necessitates an approach for measuring coverage that satisfies three requirements. It needs to be automated, interpretable and capable of discriminating between semantically distinct scenarios. Without automation, measuring coverage across large datasets is unfeasible; without interpretability and discrimination, developers cannot identify missing scenarios or understand causes of failure. Consequently, this section presents S^3C, an approach that leverages SGs to compute dataset coverage, enabling automated and interpretable analysis of dataset adequacy.

4.1.1 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. The abstraction f allows 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.

To make $cov_f(T, D)$ satisfy the three requirements defined earlier - automated, interpretable, and discriminating - we need to overcome two main issues.

The first issue 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) as detailed in Section 3.1. SG abstractions, $sg(x)$, can be computed automatically using SGGs and are interpretable because they encode semantic information. However, these are just the first steps in the input abstraction process, which allows subsequent abstractions, α, to be composed, $f = \alpha \circ sg$, to further discriminate relevant system behavior.

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 in code 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, $|\hat{A}_{id}| = 256^{(100\times100\times3)} \approx 10^{72247}$.

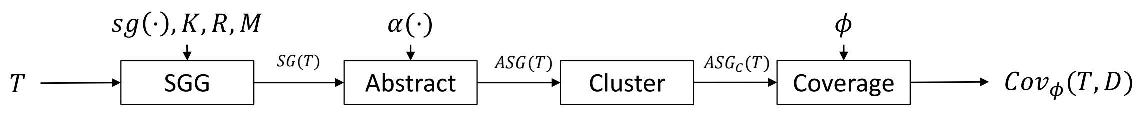


Figure 4.1: S^{3}C components, parameters, and ordering.

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.

4.1.2 Approach

The S^{3}C architecture consists of sequence of 4 components shown in Fig. 4.1 whose functionality is described next.

4.1.2.1 Scene Graph Generation

As explained in Section 3.1.2, an 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 SGG RoadScene2Vec [70] uses a forward facing camera image to generate a scene graph. A configuration file enables the user to tailor the content of the SGs to their specific ODD and AV characteristics by specifying the entity and relationship kinds, e.g., if the ODD is a rural environment, the entity list may include tractors.

4.1.2.2 Abstraction

Abstractions transform SGs to retain salient information for analysis. As explained in Section 3.1.3.3, 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.

For example, the SG vertices generated by RoadScene2Vec include an identifier attribute associated with each entity to differentiate among multiple instances of an entity class. Sometimes, such identifiers can be superfluous to the spatial distribution of entities a coverage domain is meant to capture and can be abstracted from the graph. For example, the bottom right of Figure 3.4 shows 'car_0' in lane 24 +1—the spatial relationships would be identical if it is relabelled 'car'. An *id*entifier suppressing abstraction might be defined as $\alpha_{no-id}(g) = g'$, where $g = g'$ except that $\forall v \in g.V : v.kind = car \implies g'.v.att.M[id] = \bot$. 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 $g'.V$.

4.1.2.3 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:

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

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 in 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 [162, 163]. 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. Then, the final equivalence classes are computed 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, defined as the set of SGs containing no entities other than the ego vehicle and its lane (which are always present as detailed in Section 3.1.2)—the clustering computation can be performed within practical time scales.

For example, Figure 4.2 visualizes the count and size of the equivalence classes of an abstraction, ELR, further discussed in Section 4.1.3, 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 $ASG_C(T)$

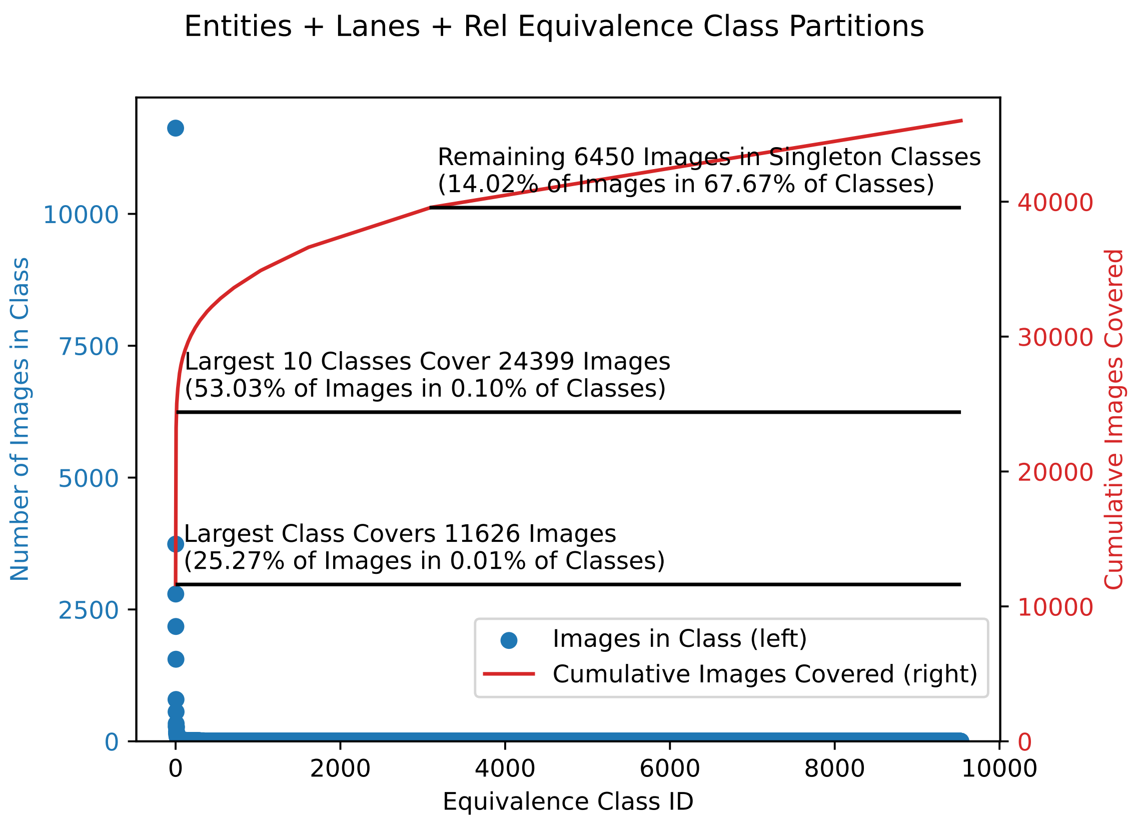


Figure 4.2: Distribution of images across scene graph equivalence classes for the ELR abstraction studied in Section 4.1.3.2

took 1047 seconds[1]. Section 4.1.4 discusses time performance for additional datasets.

4.1.2.4 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, $\phi_{\mathcal{X}} : D \mapsto Boolean$ Let A be the codomain of $\alpha \circ sg$—the set of possible abstracted scene graphs. We say that A is *interpretable* relative to $\phi_{\mathcal{X}}$ when it reflects the semantic distinctions made by $\phi_{\mathcal{X}}$ in the input domain:

$$\phi_{\mathcal{X}} \iff \forall x \in D : a = \alpha(sg(x)) \implies \phi_{\mathcal{X}}(x)$$

[1]30 threads. 32 logical core Xeon 4216, 128GB of RAM

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

We apply a form of model-based *slicing* [164] to reduce the coverage domain from $A_{\phi_{\mathcal{X}}}$ to include a single representative of each subset of $A_{\phi_{\mathcal{X}}}$ that varies only in features on which $\phi_{\mathcal{X}}$ is dependent. For each clause in the disjunctive normal form encoding of $\phi_{\mathcal{X}}$, 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 $\psi(a)$:

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

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 $\phi_{\mathcal{X}}$ 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.

For instance, consider the test specification precondition, $\phi_{\mathcal{X}}$: the closest lane to the left of ego contains a car at distance: $d_1, \ldots,$ or d_n; i.e. for an input, $i \in D, \phi_{\mathcal{X}}(i) \iff i$ 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.

4.1.2.5 Handling input sequences

The work presented so far has focused only on testing single-instant camera image inputs, which may not generalize to multi-frame inputs. We therefore present an extension to S^{3}C to handle such inputs. This extension requires modifying the clustering approach to operate on a history of scene graphs over a desired window, rather than on the scene graph from a single instant. For instance, using a window of 2 frames, the approach clusters the pair of scene graphs corresponding to the current and previous frames. We describe this modified process in more detail below.

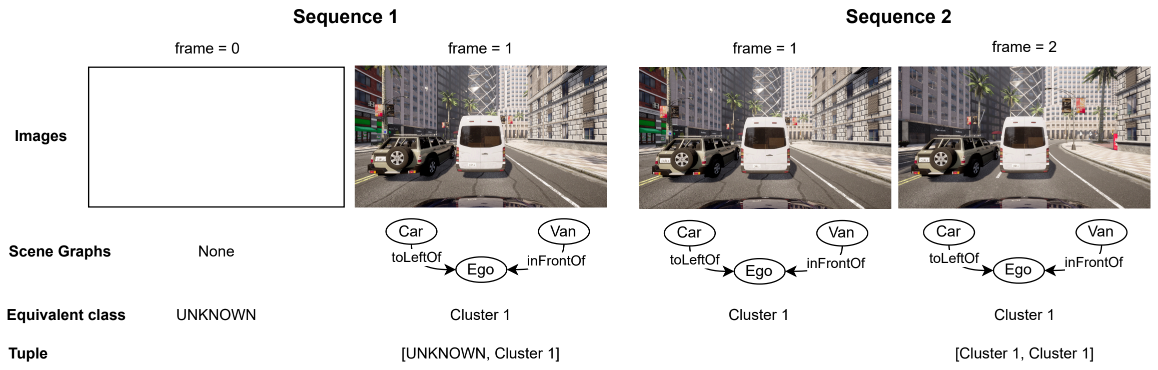


Figure 4.3: Sequences of 2 frames, including the image, its SG, the equivalent class the SG belongs to, and the tuple with the equivalent classes.

Given a window of t frames, e.g., $t = 2$ in Figure 4.3, we compute the sequence-based clustering ASC_C^t. Initially, ASG_C is used to cluster the graphs as described in Section 4.1.2.3; then each equivalence class is assigned a unique label. Next, we iterate over the frames in chronological order and, for each frame, take the most recent t frames and build a tuple of length t containing the equivalence class labels of the corresponding scene graphs. There may not exist t previous frames, e.g., at the beginning of the test or if data points were filtered out in the middle of the test - as shown in sequence 1 in Figure 4.3. In these cases, we fill in a label of `UNKNOWN` for all missing frames and consider all instances of `UNKNOWN` to be equivalent. Now, each scene graph has a t-tuple (bottom of Figure 4.3) describing the equivalence classes of the scene graphs over the previous t frames. We then re-cluster the scene graphs through the process described in Section 4.1.2.3 using the t-tuple as the basis for equivalence, i.e., two scene graphs are equivalent if they are equivalent and their previous $t-1$ graphs are also equivalent. Note that for $t = 1$ this is equivalent to the original clustering. The new clustering ASC_C^t will have at least as many equivalence classes as ASG_C, but may have many more.

4.1.2.6 Limitations

The S^{3}C approach has four main limitations. First, the quality of the analysis is fundamentally dependent on the accuracy and completeness of the SGG component. Inaccuracies in the generated SGs, such as perception failures, will propagate through the subsequent analysis steps. Second, the clustering component relies on graph isomorphism, which can be computationally intensive for large datasets or highly complex scene graphs, despite the use of practical optimizations. Third, the effectiveness of the approach is highly sensitive to the design of the abstraction function. Devising an abstraction that effectively balances discrimination and interpretability remains a manual, expert-driven task. Finally, the accuracy of the coverage metric relies on computing a tight approximation of the feasible coverage domain. While the current approach provides tight approximations for specifications with independent clauses, computing valid approximations for test specifications with complex dependencies among precondition clauses remains a challenge.

The extension for handling input sequences introduces further limitations. This process relies on clustering t-tuples of equivalence class labels. As the window size t increases, the number of distinct sequence classes can grow substantially. This may result in a highly sparse distribution, where a large fraction of the sequence classes are singletons, i.e., containing only one sequence, undermining the primary goal of clustering, which is to aggregate data into meaningful, recurring patterns. Additionally, while the individual scene graphs comprising a sequence are designed to be interpretable, the semantic meaning of a sequence of these abstracted class labels is not explicitly defined. Determining the interpretability of these temporal sequences remains an area for future work.

4.1.3 Evaluation

We empirically evaluate the effectiveness of S^3C through an experiment guided by the following three research questions[2].

- RQ#1: How effective is S^3C 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^3C to assist in the interpretation of the differences between datasets and the coverage achieved for various dataset subsets with regard to test specifications?
- RQ#3: How does extending S^3C to analyze sequences of inputs, rather than single, instantaneous inputs, impact its ability to discriminate failures? We investigate the effect of temporal information on the performance of different abstractions.

4.1.3.1 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^3C will outperform alternative approaches, and that the best instance of S^3C'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.

[2]We make our code and results available at: https://github.com/less-lab-uva/s3c

Dataset

A dataset of sensor inputs was collected by launching a pre-programmed AV in a simulated environment. We use the CARLA simulator described in Section 3.1.3. 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.

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 [165] model based on ResNet34 [166] pre-trained on ImageNet [167] 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. 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%).

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

Table 4.1: Scene graph abstractions studied. $\lvert Img \rvert$ = 46,006

Treatments

We used the CARLA SGG, described in Section 3.1.3, to generate the SGs in our evaluation allowing to control for the quality of SGs as a coverage domain[3] To evaluate the effect of the SGG-based abstraction considered, we explore five abstractions of varying specificity as outlined in Table 4.1. 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 [70] that add information about lanes and inter-entity relationships. We utilize the default inter-entity relationship configuration of RoadScene2Vec to add spatial relationships to the graph. The spatial relationships include information about front versus rear and side versus direct, giving 4 combinations: Direct Front (DF), Side Front (SF), Direct Rear (DR), Side Rear (SR). This is parameterizable by defining the threshold between these distinctions. The default parameterization uses 45 degree increments, giving each of the 4 combinations 90 degrees total, as shown in Figure 4.4.

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

[3]Section 4.1.4 provides a complementary exploration using existing SGGs on real-world datasets..

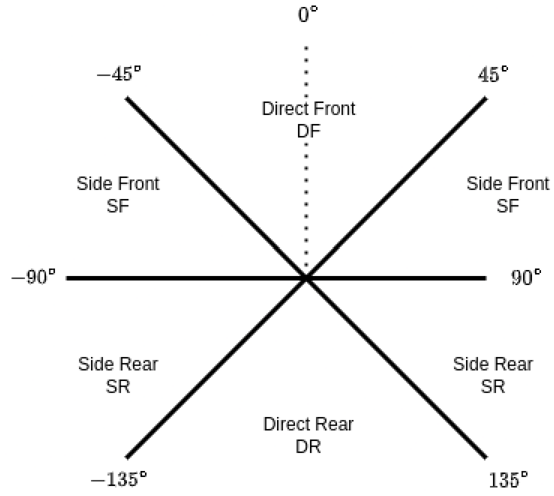


Figure 4.4: RoadScene2Vec spatial relationships.

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 [168] or from onboard sensing [169]. 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.

4.1.3.2 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.

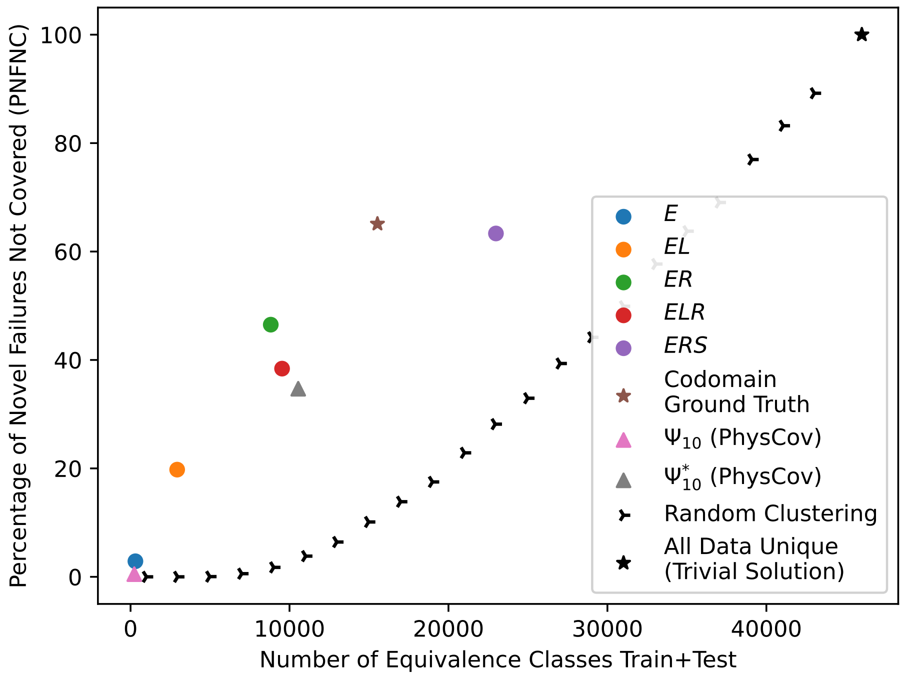


Figure 4.5: Percentage of novel test failures in an equivalence class not covered in training (PNFNC) versus count of equivalence classes under different abstractions. High percentage and low total classes is better.

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 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 Section 4.1.3.1, we instantiate 3 baselines. First, the Ψ_{10} algorithm from PhysCov [104], 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[4].

Results

As seen in Figure 4.5, 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

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

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.

4.1.3.3 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 Section 4.1.3.2 while providing a sufficiently rich semantic basis for expressing the test specifications studied in Table 4.2. This semantic richness facilitates the identification of specific scene configurations where the system fails, allowing developers to understand the nature of the scenes containing those failures. 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.

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 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.

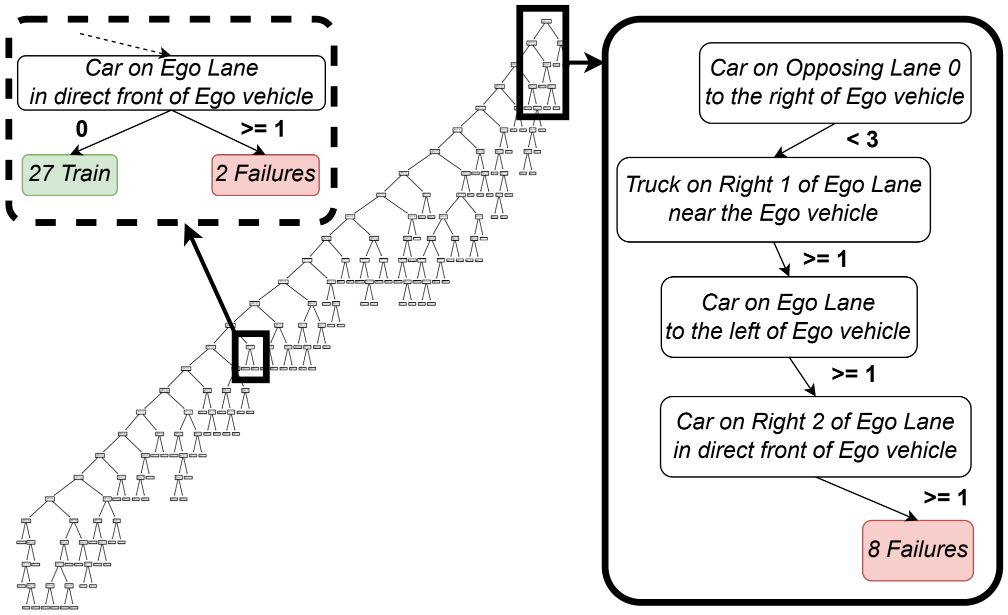

Car on Ego Lane in direct front of Ego vehicle
0
>= 1
27 Train
2 Failures
Car on Opposing Lane 0 to the right of Ego vehicle
< 3
Truck on Right 1 of Ego Lane near the Ego vehicle
>= 1
Car on Ego Lane to the left of Ego vehicle
>= 1
Car on Right 2 of Ego Lane in direct front of Ego vehicle
>= 1
8 Failures

Figure 4.6: Decision tree to characterize failures.

We find that a few spatial semantics are associated with multiple testing failures studied. For example, as shown in Figure 4.6 (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. Figure 4.7 (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 Figure 4.6 (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. Figure 4.7 (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.

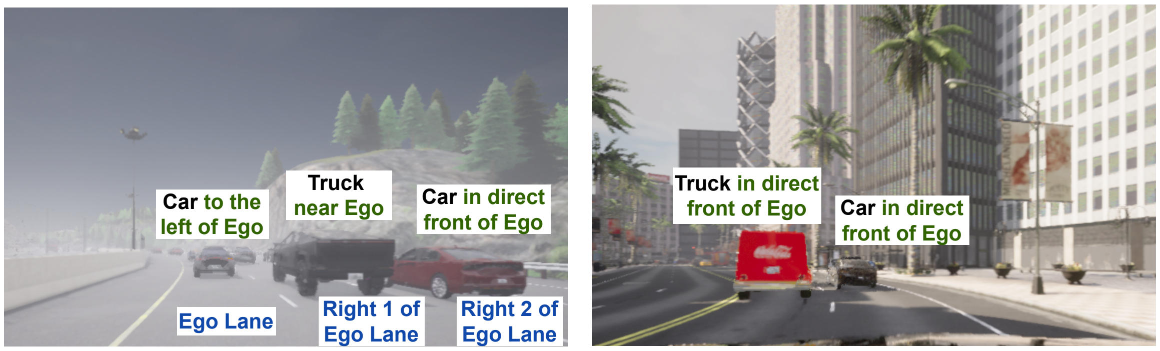


Figure 4.7: Testing images that resulted in failures.

Coverage of Preconditions

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. We selected these five preconditions to evaluate the approach by targeting distinct aspects of the driving environment: $\phi^1_{\mathcal{X}}$–$\phi^3_{\mathcal{X}}$ focus on specific, ego-centric spatial relationships critical for safety and maneuvering, while $\phi^4_{\mathcal{X}}$ and $\phi^5_{\mathcal{X}}$ capture broader scene properties regarding traffic density and lane occupancy. We selected these five preconditions to evaluate the approach by targeting distinct aspects of the driving environment: $\phi^1_{\mathcal{X}}$–$\phi^3_{\mathcal{X}}$ focus on specific, ego-centric spatial relationships critical for safety and maneuvering, while $\phi^4_{\mathcal{X}}$ and $\phi^5_{\mathcal{X}}$ capture broader scene properties regarding traffic density and lane occupancy.

Preconditions	$\lvert A\rvert$	Coverage				
		Town01	Town02	Town04	Town10HD	Full Dataset
$\boldsymbol{\phi^1_{\mathcal{X}}}$: 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%)
$\boldsymbol{\phi^2_{\mathcal{X}}}$: 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%)
$\boldsymbol{\phi^3_{\mathcal{X}}}$: 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%)
$\boldsymbol{\phi^4_{\mathcal{X}}}$: The 11 possible lanes[5] have at least a car or a truck.	22	4 (18.18%)	4 (18.18%)	22 (100.00%)	10 (45.45%)	22 (100.00%)
$\boldsymbol{\phi^5_{\mathcal{X}}}$: The 7 possible on-going lanes[5] have a car, a truck, or no vehicle.	426	3 (0.70%)	3 (0.70%)	149 (34.98%)	16 (3.76%)	163 (38.26%)

Table 4.2: Testing specifications' preconditions and their coverage.

The specifications' preconditions, their coverage domain $|A|$, and their corresponding coverage per town and for the union of all towns is shown in Table 4.2. $\phi^1_{\mathcal{X}}$ 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. $\phi^2_{\mathcal{X}}$ gets 0% coverage for Town01 and Town02 which is expected due to the absence of a left lane

[5]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).

in these towns. $\phi^3_{\mathcal{X}}$ 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 $\phi^1_{\mathcal{X}}$ and $\phi^3_{\mathcal{X}}$. On the other hand, $\phi^4_{\mathcal{X}}$ and $\phi^5_{\mathcal{X}}$ highlight the influence of the road topology. Since Town01 and Town02 only have 2 lanes, they provide a coverage of 18.18% for $\phi^4_{\mathcal{X}}$, while Town10HD with 5 lanes attains 45.45%, and Town04 with 11 lanes reaches 100% coverage. Similar findings apply to $\phi^5_{\mathcal{X}}$.

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 augmentation for addressing its deficiencies.

4.1.3.4 RQ#3: Coverage on sequences

In this section, we explore the effect of the sequence-based approach on the equivalence classes generated under the ELR, ERS, and Ψ^*_{10} abstractions for windows of length 2, 5, and 10 frames, so chosen to represent a minimal sequence and sequences of 1 and 2 seconds in duration respectively. This aims to understand the impact that increasing the amount of data considered in the sequence has on the PNFNC metric and number of equivalence classes. We denote the technique using abstraction α with a t-window as $\alpha\langle t\rangle$.

In Figure 4.8 we see that for each of the abstractions adding more information with the sequence-based approach improves the performance. However, we note that the rate of improvement is different between the abstractions; although starting from similar positions, $ELR\langle 10\rangle$ achieves a PNFNC of 62.3% in 18591 equivalence classes compared to the 68.2% of $\Psi^*_{10}\langle 10\rangle$ in 31481 equivalence classes. As such, adding timing information is proportionately much more beneficial for ELR than PhysCov. We also note that the sequence information allows $ELR\langle 10\rangle$ to achieve a similar PNFNC performance to ERS with single-frame inputs: $ELR\langle 10\rangle$ achieves a PNFNC only 1 percentage point lower

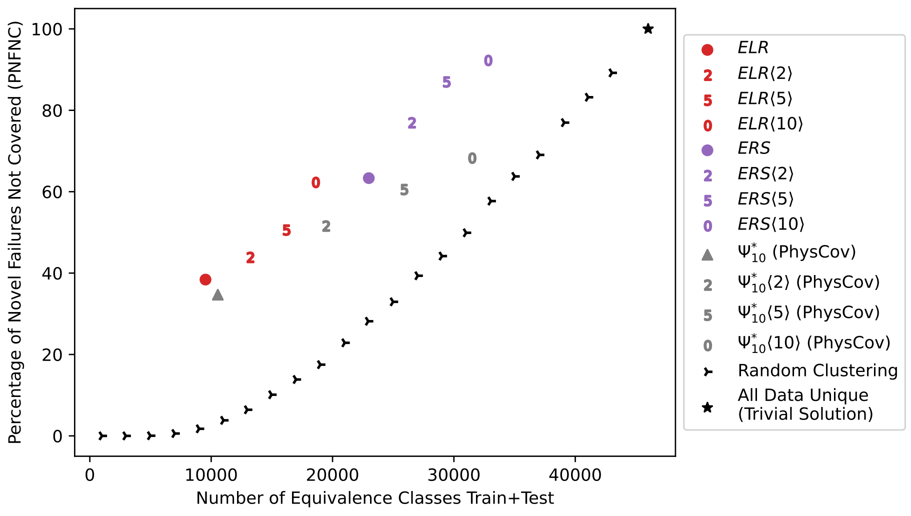


Figure 4.8: PNFNC versus count of equivalence classes for ELR, ERS, and Ψ^*_{10} with 2, 5, and 10 frame windows. High percentage and low total classes is better.

(62.3% vs 63.3%) while using 19.1% fewer equivalence classes (18591 vs 22987). Additionally, we observe that although the sequence information helps PhysCov increase the PNFNC metric, its performance relative to the random baseline decreases, and it experiences a far greater increase in the number of equivalence classes than either of the scene-graph based approaches. These results highlight the potential for improvement in S^{3}C with additional consideration of sequential information and encourage future research to this end to further enable S^{3}C's robust application to systems that leverage such information.

4.1.3.5 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. As previously detailed in Section 3.1.3.5, this evaluation relies on privileged, ground-truth information and is subject to the inherent simulation-reality gap, both of which are limitations of the chosen simulation environment. While we believe the demonstrated discrimination 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 Section 4.1.4, 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 [105, 114, 170, 171], 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 [172]. We provide an initial exploration into using S^{3}C with multi-frame inputs in Section 4.1.3.4.

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.

4.1.4 Case Study

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 4.3 of different sources, sizes, environments, and image resolutions.

Dataset	Environment	Resolution	Images		$\|ASG_C\|$		$\|Img\|/\|ASG_C\|$
			#	%	#	%	Ratio
CommaAi [173]	Highway	320x160	522,434	83%	27,837	57%	18.77
NuScenes [174]	Urban	1600x900	16,095	3%	8,879	18%	1.81
Chen [175]	Suburban	455x256	45,568	7%	8,363	17%	5.45
Udacity [176]	Highway	640x480	39,422	6%	6,251	13%	6.31
Cityscapes [177]	Urban	2048x1024	5,000	1%	4,902	10%	1.02
Union			628,519	100%	48,959	100%	12.84

Table 4.3: Equivalence Classes generated using RoadScene2Vec for open AV datasets

RoadScene2Vec suffers from 2 major problems. First, it always assumes a three-lane structure of "Right", "Middle", and "Left" as described in Section 4.1.3.1 (Treatments). Second, it 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. This process highlights that a key challenge in applying S^3C to real-world data is mitigating the types of failures observed—namely, imprecise abstractions and perception errors—by refining the abstractions used and improving the capabilities of the underlying perception components. We note that the reported performance was under RoadScene2Vec's default configuration, yielding a lower bound of performance. Better parameter tuning may eliminate some perception failures, while reducing the effects of the imprecise abstraction requires more fundamental changes.

Notwithstanding the potential weaknesses of RoadScene2Vec, we explore the scene graph based equivalence classes it generated for the datasets. Table 4.3 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.

Dataset	Person	Bicycle	Motorcycle	Bus	Car
CommaAi	0.10	0.00	0.02	0.00	2.26
NuScenes	1.53	0.07	0.04	0.18	3.01
Chen	0.14	0.00	0.00	0.01	2.31
Udacity	0.07	0.00	0.01	0.07	3.10
Cityscapes	4.04	0.97	0.15	0.16	7.65

Table 4.4: Entity statistics captured by RoadScene2Vec for open AV datasets

Table 4.4 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.

Dataset	SGG[6] (hr)	SGG per Img (s)	Abstraction+ Clustering[7] (hr)	Abstr.+Clust. per Img (s)
CommaAi	115.5	0.80	3.92	0.03
NuScenes	9.6	2.16	0.80	0.18
Chen	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

Table 4.5: Computation time for each dataset

Last, we discuss the efficiency of the S^{3}C implementation to process these datasets in Table 4.5. 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

[6]No parallelization. System with 32 logical cores, >64GB of RAM, and 4 GTX1080Ti
[7]64 threads. System with 48 logical cores and 512GB of RAM

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.

4.1.5 Summary

This section detailed S^{3}C, an approach for evaluating the adequacy of ADS training data. The evaluation showed this method can discriminate failure-inducing inputs by grouping them into classes based on SGs, while enabling the interpretation of the specific spatial distributions associated with those failures. This analysis allows developers to measure dataset coverage and identify gaps relative to safety requirements. With a method for measuring scene coverage established, the focus now shifts to ensuring the ADS learns safe behaviors. The following section introduces T4PC, which uses a property-based loss function to improve property conformance during training.

4.2 Training for Property Conformance

While ensuring a training dataset covers a wide range of scenarios is a necessary step for exposing a model to diverse driving conditions, this exposure alone is insufficient for learning safe driving behaviors. A remaining problem is compelling the ADS to consistently recognize situations where safety rules apply and to avoid actions that would violate them. Standard optimization objectives, such as minimizing prediction error against ground-truth labels, do not penalize such violations. Consequently, a DNN may learn to mimic unsafe or undesirable behaviors present in the dataset without a mechanism to steer its learning process towards property conformance. This creates a need for a method that can integrate formal property specifications into the DNN's optimization loop, teaching the model what it should not do. In this section, we present Training for Property Conformance (T4PC), an approach that addresses this challenge. T4PC integrates a custom, property-based loss function directly into the training pipeline, penalizing deviations from specified safe behaviors and using Lagrangian optimization to balance the dual objectives of achieving high accuracy on the primary driving task and ensuring conformance with safety properties.

4.2.1 Approach

To improve the property conformance of learned components, we address two key challenges. First, we develop an approach to evaluate property preconditions over high dimensional sensor inputs. Second, we develop a method for incorporating loss terms to minimize property violations while maximizing DNN accuracy. In addressing these challenges, we also propose solutions to address the issues of data sufficiency—that there is enough data for each specified property—and property consistency—that the set of properties do not conflict. Unlike S^{3}C, which focused on quantifying the coverage of existing data, the solution proposed here actively increases the number of data points satisfying property preconditions to ensure the model has enough data for training it during optimization. These methods are integrated into the T4PC approach[8].

4.2.1.1 Overview

Figure 4.9 depicts the two phases of T4PC. The property label generation phase (blue) takes a dataset, $(x, y) \in D$, and set of correctness properties, P, and enhances the dataset, D^*, prior to training. The training for specification conformance phase (green) uses the enhanced dataset and properties to train a given DNN architecture, N, and an optional set of pre-trained weights, θ, to produce an enhanced DNN, N^*, that better conforms to the properties.

Phase one (blue)

Accepts a dataset of labeled training instances, (x, y), and a set of property specifications, $\phi^i_{\mathcal{X}} \implies \phi^i_{\mathcal{Y}} \in P$, as explained in Section 3.2.1. To evaluate properties over high dimensional sensor data, each input, x, is **abstract**ed, $\alpha(x)$, and fed to a property **eval**uation component along with the associated label, y, and the properties, P. For an input, x, the pre and postconditions for all properties are

[8]We make our code and results available at: https://github.com/less-lab-uva/T4PC

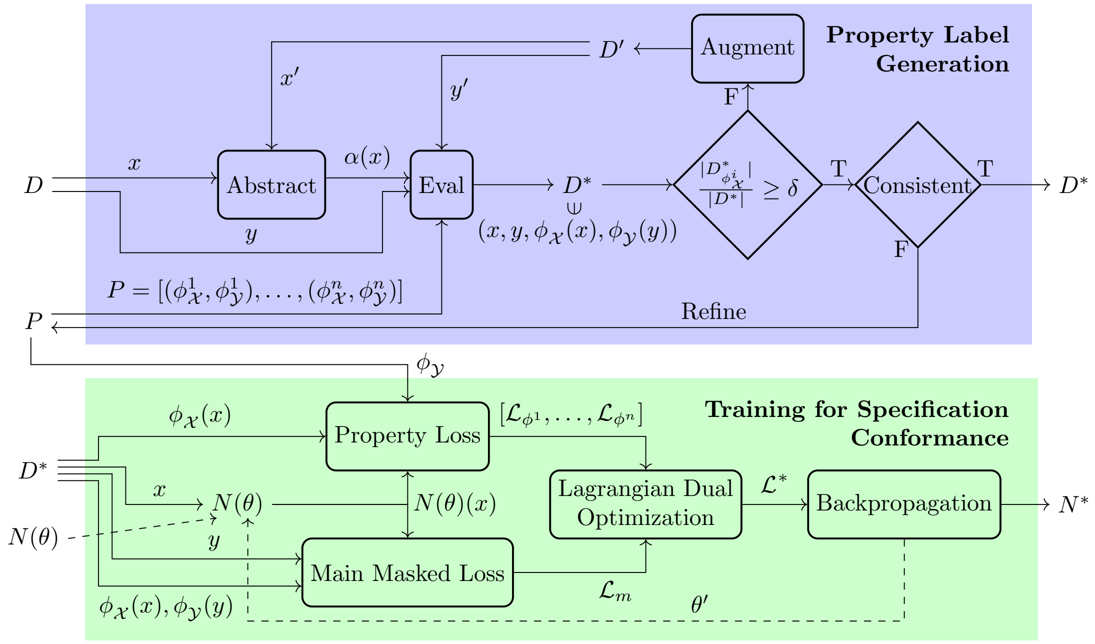

Property Label Generation
D
x
y
Abstract
$\alpha(x)$
Eval
D^*
$(x, y, \phi_{\mathcal{X}}(x), \phi_{\mathcal{Y}}(y))$
$\frac{|D^*_{\phi^i_{\mathcal{X}}}|}{|D^*|} \geq \delta$
T
F
Augment
D'
x'
y'
Consistent
T
F
D^*
$P = [(\phi^1_{\mathcal{X}}, \phi^1_{\mathcal{Y}}), \ldots, (\phi^n_{\mathcal{X}}, \phi^n_{\mathcal{Y}})]$
P
Refine
$\phi_{\mathcal{Y}}$
Training for Specification Conformance
D^*
$\phi_{\mathcal{X}}(x)$
Property Loss
$[\mathcal{L}_{\phi^1}, \ldots, \mathcal{L}_{\phi^n}]$
x
$N(\theta)$
$N(\theta)(x)$
$N(\theta)$
y
Lagrangian Dual Optimization
$\mathcal{L}^*$
Backpropagation
N^*
Main Masked Loss
$\mathcal{L}_m$
θ'
$\phi_{\mathcal{X}}(x), \phi_{\mathcal{Y}}(y)$

Figure 4.9: T4PC: phase 1 (blue) generates property labels for dataset D, and phase 2 (green) uses these labels to train a model N to conform to properties P outputting N^*, a model with improved property conformance.

evaluated to produce Boolean vectors indexed by property index, $\phi_{\mathcal{X}}(x) = [\phi^i_{\mathcal{X}}(\alpha(x)) : i \in [1, n]]$ and $\phi_{\mathcal{Y}}(x) = [\phi^i_{\mathcal{Y}}(x) : i \in [1, n]]$. The ith property is *active* for an input if its precondition is true, $\phi_{\mathcal{X}}(x)[i]$, and it *holds* for an input if $\phi_{\mathcal{X}}(x)[i] \implies \phi_{\mathcal{Y}}(y)[i]$. The output of the evaluation component is an enhanced dataset, D^*, that includes the evaluation results for pre and postconditions along with the original training instance.

To address the data sufficiency challenge, T4PC checks whether the ratio between the number of inputs meeting the precondition for each property and the total number of samples is above a parameterized threshold. We distinguish this notion of *sufficiency*—which demands a minimum number of examples per property to ensure the optimization objective is effective—from the *adequacy* measured by S^{3}C, which evaluates the semantic coverage across the domain. If this threshold is not met, implying the data is insufficient for learning, the dataset is **augment**ed to generate additional training instances, $(x', y') \in D'$, which are then abstracted, evaluated, and added to D^*.

If sufficient data is available for all properties, then T4PC checks that the properties are **consis-**

tent relative to the dataset. This means that if an input satisfies the preconditions of two properties, then it must be possible to satisfy both of their postconditions. Resolving inconsistency among properties requires that the developer **refine** the specifications. The enhanced, augmented, and sufficient dataset, D^*, for a set of consistent properties, P, is then used as input for the second phase.

Phase two (green)

Takes the dataset, D^*, the properties, P, and a deep neural network, N, as input. To accommodate the application of T4PC to pre-trained networks, the set of network parameters, θ, can be provided as input as well. When θ is understood from the context we write $N(\theta)$ as N.

T4PC splits training for a data item, $(x, y, \phi_{\mathcal{X}}(x), \phi_{\mathcal{Y}}(y)) \in D^*$, into the calculation of two loss terms. The **main masked loss** term which typically consists of computing $\mathcal{L}_m = \|N(x) - y\|$, but in T4PC also incorporates a check of $\phi_{\mathcal{X}}(x) \implies \phi_{\mathcal{Y}}(y)$ to *mask* the loss of any property violations present in the training data. The **property loss** term computes, for each property i, the distance from the DNN prediction to the nearest point satisfying the postcondition, $\mathcal{L}_{\phi^i} = \|N(x) - \phi^i_{\mathcal{Y}}\|$.

To appropriately blend the $n+1$ loss terms, T4PC uses the Lagrangian dual optimization [156]. In this process, the property losses, $[\mathcal{L}_{\phi^1}, \dots, \mathcal{L}_{\phi^n}]$, are combined with the main loss, $\mathcal{L}$, to produce a single loss, $\mathcal{L}^*$, that aims to balance the performance of the DNN—matching training labels—and conformance with specifications. This loss term is used to update the DNN parameters, θ', through backpropagation. Training iterations continue for a specified number of epochs at which point the final DNN, N^*, is produced. By considering the abstracted sensor inputs during training time, T4PC produces N^*, with the aim of achieving the original training goal and also improved property conformance. A more detailed description of T4PC follows.

4.2.1.2 Property Label Generation

The objective of this phase is to make a given dataset amenable for training a DNN for property conformance.

Abstract Inputs

To enable the evaluation of preconditions over inputs with high-dimensional sensor inputs such as

Prop (Rule)	English description	$\phi_{\mathcal{X}}$	$\phi_{\mathcal{Y}}$
ϕ_1 (46.2-816)	If ego has an entity within 10m in front and in the same lane, then ego should stop.	$ego.dist(\leq, 10) \cap ego.on.\hat{}\ on \cap ego.infront \neq \emptyset$	$N(x).acc \leq$ -0.25
ϕ_2 (46.2-833)	If ego lane traffic light is red/yellow, then ego should stop.	$ego.controlled.T_{light}.color \in \{red, yellow\}$	$N(x).acc \leq$ -0.25
ϕ_3 (46.2-888)	If ego has no entity within 25m in front and in the same lane, there is no red or yellow traffic light, and there is no stop sign, then ego should accelerate.	$ego.controlled.stopsign = \emptyset \wedge$ $ego.controlled.T_{light}.color \notin \{red, yellow\} \wedge$ $ego.dist(\leq, 25) \cap ego.on.\hat{}\ on \cap ego.infront = \emptyset$	$N(x).acc \geq 0.25$
ϕ_4 (46.2-833)	If there is a green traffic light and nothing in front of ego in the same lane within 10m, then ego should accelerate.	$ego.controlled.T_{light}.color = green \wedge$ $ego.dist(\leq, 10) \cap ego.on.\hat{}\ on \cap ego.infront = \emptyset$	$N(x).acc \geq 0.25$
ϕ_5 (46.2-802)	If ego is in the rightmost lane and not in a junction, then ego should not steer to the right.	$ego.on.right = \emptyset \wedge ego.junction = \emptyset$	$N(x).steer < 0.07$
ϕ_6 (46.2-804)	If ego is in the leftmost lane and not in a junction, then ego should not steer to the left.	$ego.on.left = \emptyset \wedge ego.junction = \emptyset$	$N(x).steer >$ -0.07
ϕ_7 (46.2-821)	If ego is moving and there is a stop sign affecting it within 10m, then ego should stop.	$ego.speed \geq 0.1 \wedge$ $ego.dist(\leq, 10) \cap ego.controlled.stopsign \neq \emptyset$	$N(x).acc \leq$ -1.00
ϕ_8 (46.2-821)	If ego is stopped and there is a stop sign affecting it within 10m and there is nothing in front in the same lane within 7m, then ego should accelerate.	$ego.speed < 0.1 \wedge$ $ego.dist(\leq, 10) \cap ego.controlled.stopsign \neq \emptyset \wedge$ $ego.dist(\leq, 7) \cap ego.on.\hat{}\ on \cap ego.infront = \emptyset$	$N(x).acc \geq 0.75$

Table 4.6: Properties with preconditions ($\phi_{\mathcal{X}}$) in RFOL over SGs and postconditions ($\phi_{\mathcal{Y}}$) as DNN output constraints.

image data, T4PC uses scene graphs as an abstraction of sensor inputs, $\alpha = sg$, as described in Section 4.1.2.2. This involves identifying the kinds of entities and relations mentioned across a set of properties and using those to define the vertex and edge sets of the SG. We selected the set of properties shown in Table 4.6 to provide a representative sample of safety-critical driving rules derived from the Virginia Driving Code [178]. These rules were chosen to cover diverse aspects of driving, including longitudinal control, such as stopping for red lights or obstacles (ϕ_1-ϕ_4, ϕ_7, ϕ_8), and lateral control, such as lane keeping (ϕ_5, ϕ_6).

For example, given the set of preconditions, $\phi_{\mathcal{X}}$, for the properties shown in Table 4.6, the set of relations in an SG must include: $dist(op, d)$, relations indexed by relational operator, op, and distance, d; and *on*, *infront*, *light*, *color*, *speed*, *stopsign*, *right*, *junction*, and *left*. We note that *color* and *speed* define attributes—a special class of relations whose image is a singleton set.

Generating such SG permits the evaluation of preconditions over $\alpha(x)$ as explained in Section 3.2.1.

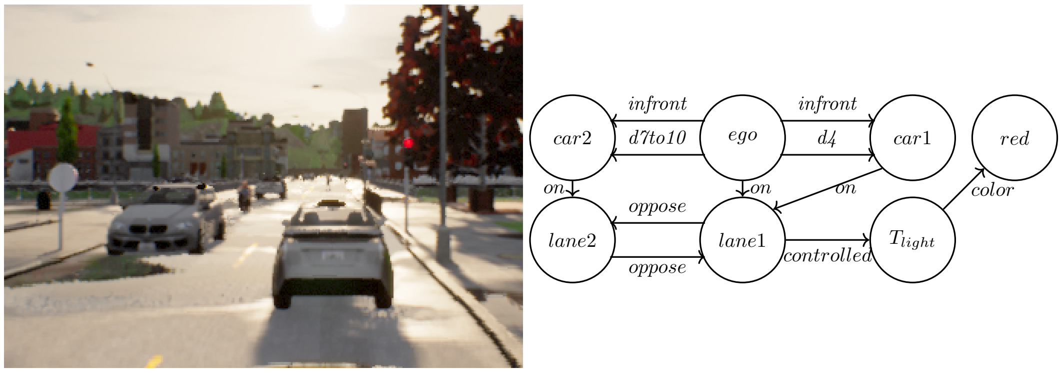


Figure 4.10: An image (left) and a portion of its scene graph (right).

Figure 4.10 depicts an image and its associated scene graph which satisfies the preconditions of ϕ_1 and ϕ_2. In this SG, the family of distance relations includes: $d4$ meaning that the distance is less than or equal to 4m, and $d7to10$ meaning the distance is between 7m and 10m, both of these are included in the relations defined by $dist(\leq, 10)$.

Evaluate Properties

Evaluating $\phi_{\mathcal{X}}$ involves compiling the precondition to a traversal of the SG; e.g., for $\phi^1_{\mathcal{X}}$ on the SG from Figure 4.10, this determines that the set of vehicles that occupy the lane of the ego vehicle, specified as $ego.on.\hat{}\,on$, includes a vehicle, $car1$, that lies in the image of relation $dist(\leq, 10)$ relative to ego. For $\phi^2_{\mathcal{X}}$, the SG traversal determines that the color of the light governing the ego vehicle lane, $ego.controlled.T_{light}.color$, is red, so this precondition is also satisfied. The Boolean evaluations of all preconditions on $\alpha(x)$ are stored, $\phi_{\mathcal{X}}(x)$, for use in subsequent processing.

Postconditions, $\phi_{\mathcal{Y}}$, are formulated over DNN outputs, $N(x)$, which for a training instance is the label, y. In this work, postconditions are conjunctions of axis-aligned linear constraints which define *hyper-rectangular* constraints in the DNN output space. Given a training instance, (x, y), we can evaluate the postcondition $\phi_{\mathcal{Y}}(y)$ by directly evaluating the linear constraints. For example, both ϕ_1 and ϕ_2 have the same postcondition, $N(x).acc \leq -0.25$, which requires the acceleration output of the DNN to indicate deceleration. The Boolean outcomes of evaluating all postconditions on labels, y, are stored, $\phi_{\mathcal{Y}}(y)$, for use in subsequent processing.

We store pre and postconditions separately, since this allows us to efficiently determine whether a training instance satisfies a precondition, $\phi_{\mathcal{X}}(x)$, and property, $\phi_{\mathcal{X}}(x) \implies \phi_{\mathcal{Y}}(y)$.

Data augmentation

This module addresses the need for data sufficiency relative to property preconditions. A dataset must have enough training samples that satisfy each property precondition, $\phi^i_{\mathcal{X}}$, in order for DNN training to be able to learn to conform to the property. Let

$$D^*_{\phi^i_{\mathcal{X}}} = \{x : x \in D^* \wedge \phi_{\mathcal{X}}(x)[i]\}$$

be the input samples that satisfy ϕ^i's precondition. A dataset is *insufficient* for a property, ϕ^i, if the ratio of training inputs that satisfy its precondition is below a threshold, $\frac{|D^*_{\phi^i_{\mathcal{X}}}|}{|D^*|} < \delta$. If this inequality holds, then we must generate at least $t_i = \left\lceil \frac{\delta|D^*| - |D^*_{\phi^i_{\mathcal{X}}}|}{1-\delta} \right\rceil$ new inputs for D^* through augmentation that satisfy the precondition; this is the least integer value added to the numerator and denominator of the left side of the threshold inequality that guarantees it will be false.

To minimize the impact of augmentation on the data distribution, we wish to transform elements of the dataset that are the closest to satisfying $\phi^i_{\mathcal{X}}$. Since preconditions are formulated over α, we define "distance to satisfaction" as:

$$\|x - \phi^i_{\mathcal{X}}\| = \min_{x' \in D^*_{\phi^i_{\mathcal{X}}}} \|\alpha(x) - \alpha(x')\|$$

as the minimum distance from x to a training input x' that satisfies the precondition. This provides a pool of transformation candidates, $C = \{(x, d) : x \in D^* \wedge d = \|x - \phi^i_{\mathcal{X}}\|\}$. We then select t_i values to augment, $D' \subseteq C$, where $|D'| = t_i$, and $\forall (x', d') \in D' : \forall (x, d) \in C - D' : d' \leq d$. As we discuss next, it may be possible to augment the same data point multiple ways; if so, $|D'| \leq t_i$.

We augment the dataset using metamorphic transformations. A metamorphic transformation consists of a pair of functions $f : X \rightarrow X$ and $g : Y \rightarrow Y$ that modify the input and output, respectively, while preserving a consistency relationship such that for any correct implementation, $h : X \rightarrow Y$, the transformation ensures that $\forall x, y : g(y) = h(f(x))$. Such f and g must be carefully crafted by the developer with respect to the requirements to ensure this validity holds.

To illustrate, consider a driving dataset where inputs, $x = (i, s)$, consist of an image, i, and the speed, s, of ego vehicle, with outputs, y, defining acceleration. A property might specify that: *"if there is a car stopping in front of the ego vehicle and ego vehicle speed is greater than 5 km/h, then ego vehicle should stop"*. An input x containing an image that shows a car with its brake lights illuminated in front of the ego vehicle and a speed of 2 km/h, does not satisfy the precondition of the property because the speed is less than 5 km/h. This input could be augmented using function $f(x) = (i, rand(minSpeed, maxSpeed))$ that (potentially repeatedly) chooses a speed within the speed range, and $g(y) = y$ to preserve the output. If a data point had a speed of 10km/h, but the image lacked the car in front of ego, a function $f(x) = (addCarFrontEgo(i), s)$ could be used as proposed by Woodlief et al. [179].

Check Consistency

T4PC is designed to train for conformance with a set of properties. It is possible, however, that two properties are inconsistent. For example, if we modified the precondition of ϕ_1 to use an equality rather than the disequality, then this precondition would subsume the precondition of ϕ_4, which additionally constrains the color of the light governing the ego vehicle lane. Having such properties would be problematic because $\neg\exists y : \phi^1_{\mathcal{Y}}(y) \wedge \phi^4_{\mathcal{Y}}(y)$, which means that it would be impossible to satisfy both of these properties. T4PC requires that the set of properties, P, be defined so as to avoid such situations.

Given a universe of scene graphs, U, one can determine whether the preconditions of two properties, ϕ and ϕ', overlap by evaluating $\exists u \in U : \phi_{\mathcal{X}}(u) \wedge \phi'_{\mathcal{X}}(u)$. Given a distribution, $\mathcal{D}$, of sensor inputs, if U were guaranteed to subsume all scene graphs that could occur, $\forall x \sim \mathcal{D} : \alpha(x) \in U$, then it would be sound to use such an approach. Determining such a U is challenging, moreover existing approaches to reasoning about RFOL do not scale to graphs of the size that we have observed in realistic driving scenes [180].

We approximate the above check by computing the co-satisfaction of precondition pairs evaluated over the dataset:

$$O = \{(i,j) : 1 \leq i < j \leq n \land x \in D^* \land \phi_{\mathcal{X}}(x)[i] \land \phi_{\mathcal{X}}(x)[j]\}$$

A set of properties is *consistent* if $\forall (i,j) \in O : \phi^i_{\mathcal{Y}} \land \phi^j_{\mathcal{Y}}$. For a general class of postconditions, including the hyper-rectangular constraints used in this work but generalizing well-beyond that, we can encode the check for consistency as an SMT problem in the theory of linear real arithmetic. Since there are at most $\frac{n^2}{2}$ pairs of matching preconditions and postconditions are not overly complex, this is a very efficient check; for the properties in Table 4.6 the total time for all checks is less than 0.01 seconds. Inconsistent pairs of properties are reported to the user who must refine them and restart T4PC.

We note that the lack of soundness of dataset consistency is not a problem for T4PC. This is because the training process, described below, only assumes dataset consistency in computing property loss.

4.2.1.3 Training for specification conformance

The purpose of this step is to use our enhanced dataset D^* to train a DNN for specification conformance by incorporating the defined properties into the optimization.

Main Masked Loss

When a training instance, (x, y), violates a specification, $\exists i : \neg(\phi_{\mathcal{X}}(x)[i] \implies \phi_{\mathcal{Y}}(y)[i])$, there are several possible solution strategies. For example, one could provide this feedback to the DNN developer and ask them to refine the training instance manually, but this could be expensive for developers. We adopt a more permissive approach that allows for violating training data, but masks the main loss term—since it may bias training away from property conformance—and effectively uses the input, x, to perform a kind of weak-supervision based on property loss.

For example, consider a property *"if there is a red light governing the ego lane, the ego vehicle should stop"*, and a data point for which the input image contains a red light, but its label says that ego should accelerate instead of stop. In this case, the approach will mask the main loss for that data point since the ground truth label for that input is wrong with respect to the property. This step is important because we do not want the DNN to learn behavior that does not conform to the specified properties.

Given a deep neural network prediction $N(x)$, a data label y, and property labels $\phi_{\mathcal{X}}(x), \phi_{\mathcal{Y}}(y)$, the main masked loss $\mathcal{L}$ is defined:

$$\mathcal{L}_m(N(x), y) = \begin{cases} 0 & \text{if } \exists i : \phi_{\mathcal{X}}(x)[i] \wedge \neg\phi_{\mathcal{Y}}(y)[i] \\ l(N(x), y) & \text{otherwise} \end{cases}$$

where l is a traditional loss function, such as, MSE, MAE, or cross-entropy. If any property is violated by the training pair, (x, y), then we say that the main loss is *masked*.

Property Loss

When properties are violated, T4PC biases training towards their satisfaction. It takes in the DNN predictions, $N(x)$, the precondition labels $\phi^i_{\mathcal{X}}(x)$ and the postcondition labels $\phi^i_{\mathcal{Y}}(y)$ for each property, and computes a loss for each property, $\mathcal{L}_{\phi^i}$. In this section, we define property loss for a general class of postconditions containing multiple output variables expressed as hyper-rectangular (HR) constraints in the output space of a DNN. This class is sufficient for expressing a range of properties like the ones defined in Table 4.6.

Let N's output space be $\mathbb{R}^m$ and for $y \in \mathbb{R}^m$ let $y[i]$ denote the coordinate value of its ith dimension. A set of m intervals, $\mathcal{I} = \{[l_i, u_i] : i \in [1, m]\}$, forms the basis for defining an HR postcondition, $\phi_y = \prod_{[l,u]\in\mathcal{I}}[l, u]$. Let $\phi_y[i]$ denote the interval for the ith dimension, $[l_i, u_i]$. The distance from a scalar value, v, to a closed interval is:

$$\|v - [l, u])\| = \begin{cases} 0 & \text{if } l \leq v \wedge v \leq u \\ \min(\|l - v\|, \|v - u\|) & \text{otherwise} \end{cases}$$

Lifting this distance to the vector of DNN outputs and associated postcondition intervals gives the property loss:

$$\mathcal{L}_{\phi^i}(N(x)) = \begin{cases} \|N(x) - \phi^i_{\mathcal{Y}}\| & \text{if } \phi_{\mathcal{X}}(x)[i] \wedge \neg\phi_{\mathcal{Y}}(N(x))[i] \\ 0 & \text{otherwise} \end{cases}$$

4.2.1.4 Lagrangian Optimization

As depicted in Figure 4.11, for each property that is violated a separate loss term is computed, e.g., $\mathcal{L}_{\phi^1}$ and $\mathcal{L}_{\phi^2}$. These losses are combined in order to bias training towards conformance of all properties.

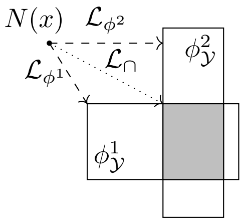


Figure 4.11: Property loss (- -), combined loss (. .)

This combined loss, $\mathcal{L}_{\cap}$ in the figure, approximates the distance to the intersection of the active post-conditions as depicted in the grey region of the figure. To blend the main masked loss and property loss terms without requiring parameter tuning, T4PC leverages Lagrangian dual optimization [156] to learn the blending hyperparameters, λ_i. Our problem's loss in this approach is:

$$\mathcal{L}^*(N(x), y) = \mathcal{L}_m(N(x), y) + \sum_{i=1}^{|P|} \lambda_i \mathcal{L}_{\phi^i}(N(x))$$

The λ_i are initialized to 0 and updated after every training epoch by incorporating the property-specific loss:

$$\lambda_i^{k+1} = \lambda_i^k + \rho_i \sum_{(x,\ldots)\in D^*} \mathcal{L}_{\phi^i}(N(x))$$

The hyperparameter ρ_i is the Lagrangian update step which can be set for each property to control the rate of change of λ_i. Property loss weights are initially zero, $\lambda_i^0 = 0$. If a property, ϕ_i, is violated then its weight, λ_i, is updated in proportion to the cumulative loss within current training epoch moderated by ρ_i, which is a small number in the range of 10^{-1}. This allows the optimization process to adapt during training to determine how to weight property loss—to achieve property conformance—while avoiding overweighting it—which might reduce DNN accuracy.

4.2.1.5 Limitations

T4PC enables training a DNN to improve property conformance. However, the approach for T4PC is limited in the expressiveness of the properties that can be leveraged for training.

First, regarding the shape of the allowable constraints, the current implementation focuses on Hyper-Rectangular (HR) constraints. We note that further generalization of postconditions, e.g., to disjunctions of HR postconditions, and to polyhedral postconditions, could be valuable. For example, handling disjunctions would allow for collision avoidance specifications like: *"if ego's speed is more than 10mph and a vehicle is 4m in front, and no vehicles are to the left or right, then the ego steering should be aggressively left or right"*, whereas the current approach requires a single convex region. Similarly, a quadrotor drone requiring an evasive maneuver illustrates the need for disjunctive constraints. To avoid an obstacle, the drone might need to either increase thrust in the front motors to pitch backward or increase thrust in the left motors to roll right. These alternative actuation patterns constitute disjoint regions in the control space, which the current approach cannot represent with a single convex constraint.

Second, to apply the property loss regime specified in Section 4.2.1.3 (Property Loss), the property must be specified over a single input-output pair describing the expected behavior in response to each input. For example, a property defined in SGL could say *"if there is a green light, acceleration must be positive"* as this is evaluated over a single input (whether there is a green light in the abstract input representation) and a single output (whether the resultant acceleration is positive). This allows T4PC to assign the attendant property loss to each input to guide the training through backpropagation. However, autonomous vehicles are bound by richer temporal properties that span multiple inputs and outputs; for example, *"the vehicle must stop at the stop sign before continuing"*. This property does not describe the expected output of the DNN at any particular time step—whether the vehicle should accelerate or decelerate depends on whether the vehicle has already stopped or not. While we previously discussed the use of LTL_f for expressing such properties in Section 3.2, integrating LTL_f into this training loop presents a distinct challenge. Evaluating an LTL_f property requires analyzing sequences of inputs, whereas the supervised learning paradigm utilized in T4PC calculates gradients based on individual, independent inputs. Consequently, translating a temporal violation observed over a sequence into a differentiable loss signal for a specific input is not addressed in the loss formulation of T4PC. Although Section 4.2.3 demonstrates how T4PC can approximate such properties during training to improve conformance at runtime, future work should investigate methods to expand T4PC to directly leverage these richer temporal properties.

4.2.2 Evaluation

To study T4PC's ability to increase property conformance of a DNN we set the following research questions:

- RQ#1: How effective is T4PC at training a DNN from scratch to conform to varying numbers of properties? We explore T4PC's performance when applied to train a series of models towards the conformance of property sets of different sizes.
- RQ#2: How efficient is T4PC in training a DNN to conform to multiple properties? We explore how the number of properties targeted affects the training time, as well as the trade-offs between training time and performance gains when training for different periods.
- RQ#3: How effective is T4PC at fine-tuning a DNN to conform to single properties?

We defer studying the treatment combination of fine-tuning for multiple properties to the case study in Section 4.2.3. To answer these research questions, we first introduce the *dataset* used for training and evaluation, followed by the *navigation DNN* used in our experiment. Second, we present a detailed description of the *properties* studied and describe our *experimental design* for both training from scratch and fine-tuning. Finally, we outline the *metrics* employed to assess T4PC's effectiveness in achieving property conformance.

4.2.2.1 Setup

Dataset

To ensure high-quality data collection, we utilized TCP [181], an end-to-end AV system (that we later study as a whole in Section 4.2.3), ranked number 3 as of June 2022 in the CARLA leaderboard challenge[9] , which evaluates the driving proficiency of autonomous agents in realistic traffic scenarios. We collected a dataset of 400,487 frames each containing an image, steering angle, acceleration, and scene graph using the CARLA simulator described in Section 3.1.3. Following the procedure defined by the TCP system, we collect data from 6 CARLA environments, also known as "Towns", that cover a range of driving landscapes, from urban to rural, including single and multi-lane roads, and various weather and lighting conditions. For each town, the TCP procedure executes many routes, where each route contains a series of waypoints that the ego vehicle must follow until reaching the destination. Depending on the town, between 321 and 480 routes are collected. The entire dataset, over 137 GB, is available to download in our repository.

During the data collection phase, we used the CarlaSGG introduced in Section 3.1.3 to generate ground-truth scene graphs (our abstraction α)[10], adopting the SG parameterization used in Section 3.2 and Section 4.1. We use ground-truth SGs to enable analysis of T4PC independent of noise or faults in scene graph generators over sensor data; however, in Chapter 5, we explore the usage of SGGs to generate SGs from real and simulated images without relying on privileged information and the results are encouraging for the future application of T4PC in practice.

AV Navigation DNN

We focus on a DNN that can be used by an AV to navigate in an environment. To facilitate experimentation, we trained our own DNN so we could manipulate the properties that we optimize for and use different data. Note that Section 4.2.3 complements this controlled setup by applying T4PC to two real systems in a simulated environment. The AV will capture images through its front camera and feed them to the DNN; the DNN will output a steering angle and acceleration

[9]The ranking was hosted at `https://paperswithcode.com/sota/autonomous-driving-on-carla-leaderboard`, which is no longer available.

[10]Data from 2 out of 8 towns were not collected due to incompatibilities between the CARLA versions used by TCP and by our instrumentation plugin.

signal that the AV will use to move around. Both DNN outputs are normalized to the range $[-1, 1]$. For acceleration, a value of -1 is the strongest signal for braking, while a value of 1 is the strongest signal for accelerating. For steering angle, these values represent a range between -70 and 70 degrees. Similar to the DNN used in Section 4.1.3, we leverage a pre-trained ResNet34 backbone since it is used by the top 3 ADS [182, 181, 183] in CARLA leaderboard, though with a modified output layer to predict acceleration and steering angle.

Properties

We define six safe driving properties for this experiment. Their English description and logic formula for their pre and postconditions are shown in the first part of Table 4.6 (ϕ_1–ϕ_6). We do not include ϕ_7 and ϕ_8 as their preconditions require the ego vehicle speed, and the AV navigation DNN used in this experiment does not take that as input (we consider them in Section 4.2.3). The first 4 properties state a postcondition in terms of the model's acceleration output, while the following 2 properties state a postcondition in terms of the model's steering angle output. Among the 4 acceleration properties, ϕ_1 and ϕ_2 expect a negative acceleration as they enforce a stop action, while ϕ_3 and ϕ_4 expect a positive acceleration.

Experimental Design

We perform three distinct experiments to answer each research question. For each question we train two types of DNNs: base DNNs ($\mathcal{B}$) that only include the main loss function computed from the data labels, and treatment DNNs ($\mathcal{M}$) using T4PC that also account for property conformance. Since we do not explore augmentation in this experiment, we have no metamorphic functions to be defined. All the DNNs, $\mathcal{B}$ and $\mathcal{M}$, throughout the three research questions are based on the architecture described in Section 4.2.2.1 (AV Navigation DNN), trained for 15 epochs[11], using batches of 256 images, with the same dataset. We use a Stochastic Gradient Descent (SGD) optimizer, a constant learning rate (lr) of 10^{-4}, and a ρ for the property losses of 0.1. For training every DNN, we used 8 CPU cores, 128G of RAM, and 1 A100 GPU.

To account for the variation across towns and routes, we implemented leave-one-out cross-validation, leading to 6 splits with 5 towns used for training and validation, and 1 town used for

[11]We show why 15 epochs is adequate as part of Section 4.2.2.4.

testing. We train and evaluate a DNN for each split, leading to 6 DNNs per type. The metrics are reported on the aggregate of the test results obtained from each split.

Metrics

We define 2 types of metrics: *violation improvement*, and *loss improvement*, using percentages. These metrics represent the difference between the property violations (V_{imp}), the steering angle (L^s_{imp}), and acceleration (L^a_{imp}) loss of the baseline, $\mathcal{B}$, and the DNN that uses property loss, $\mathcal{M}$. For each of the metrics, a higher score indicates an improvement of performance, with a value of 100.00% meaning that $\mathcal{M}$ has removed all violations present in $\mathcal{B}$. Higher values of V_{imp} indicate that T4PC is effective at improving property conformance. Note that while higher values of L^{type}_{imp} are better, a 0.00% would indicate that T4PC did not decrease the performance of the baseline DNN with respect to the original loss. Formally:

$$V_{imp} = \frac{\sum_{i=0}^{|Splits|} v(B^i) - \sum_{i=0}^{|Splits|} v(M^i)}{\sum_{i=0}^{|Splits|} v(B^i)} \times 100$$

$$L^{type}_{imp} = \frac{\sum_{i=0}^{|Splits|} \mathcal{L}_{type}(B^i) - \sum_{i=0}^{|Splits|} \mathcal{L}_{type}(M^i)}{\sum_{i=0}^{|Splits|} \mathcal{L}_{type}(B^i)} \times 100$$

Where v is a function to obtain the number of violations for a given DNN. To aggregate the results from the 6 split cross-validation, we sum the number of violations/loss across the splits. These sums are used for computing V_{imp}, L^s_{imp}, and L^a_{imp} respectively.

4.2.2.2 RQ#1: Training towards property conformance

Setup specific to experiment

We train 6 baseline $\mathcal{B}$ DNNs (one per split). We also train with T4PC for each of the 6 splits using varying combinations of the first 6 properties in Table 4.6. This results in varying numbers of $\mathcal{M}$ DNNs, based on combinations of the 6 properties. Specifically, choosing 1, 2, 4, or all 6 properties yields $\binom{6}{1} = 6$, $\binom{6}{2} = 15$, $\binom{6}{4} = 15$, and $\binom{6}{6} = 1$ combinations, respectively. Each combination is evaluated across 6 data splits, resulting in a total of 36, 90, 90, and 6 $\mathcal{M}$ DNNs, respectively.

Results

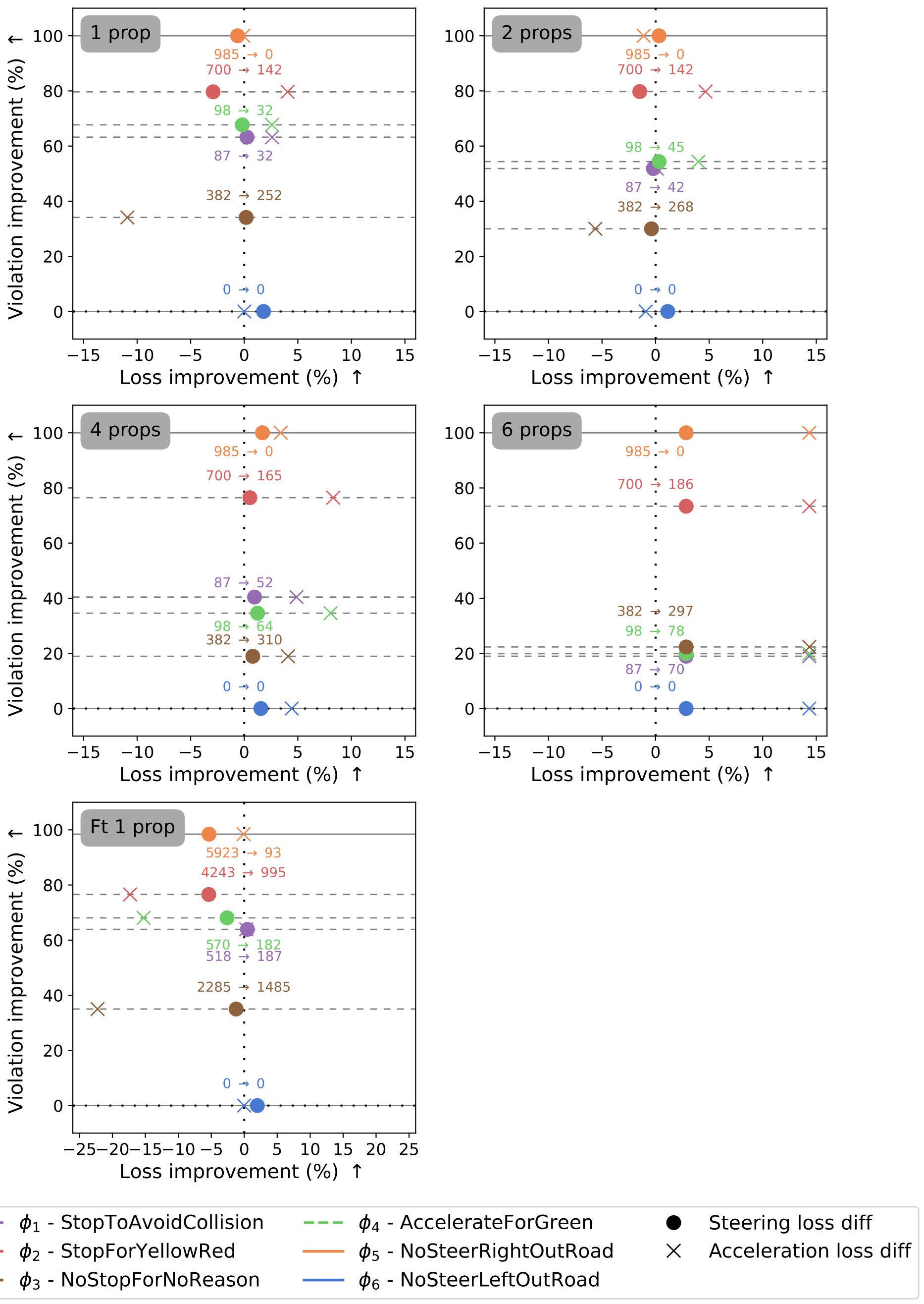


Figure 4.12: Experiment results when training from scratch with 1, 2, 4, and 6 properties, and fine-tuning with 1 property.

The first four subfigures of Figure 4.12 show the results for the treatments (instances of 1 prop, 2 props[12], 4 props[12], and 6 props[13]). The x-axis measures the main loss improvement for steering (●) and acceleration ($\times$). The y-axis measures the violation improvement where 0.00% means that $\mathcal{B}$ and $\mathcal{M}$ cause the same number of violations, and a value of 100.00% means that, different from $\mathcal{B}$, $\mathcal{M}$ does not violate the property. Negative values in either axis means that DNN $\mathcal{M}$ has either a higher loss or more violations than its counterpart $\mathcal{B}$. Ideally, $\mathcal{M}$ will result in observations at the top (violation improvement > 0) and center-right (loss improvement ≥ 0) of each figure—reducing violations but retaining the original performance of $\mathcal{B}$. To ease the reading of Figure 4.12, we used dashed lines to represent acceleration properties and solid lines to represent steering angle properties, and we included the total number of violations from $\mathcal{B}$ and $\mathcal{M}$ respectively.

We first focus on training for one property. Starting from the top of the graph in Figure 4.12 (1 prop), when T4PC is applied to conform to ϕ_5, $\mathcal{M}$ achieves a 100.00% violation reduction over $\mathcal{B}$ (from 5,912 violations to 0), meaning that it was able to learn not to violate the property unlike the base DNN. Applying T4PC to the first four properties (acceleration properties), shows a range of gains from 79.66% to 34.10%, depending in part on the DNN's capability to identify the property precondition features. For example, features such as detecting traffic light by color (ϕ_2, ϕ_3, ϕ_4) are readily identified by models like Detectron2 [39]. In contrast, distance features like entities within X meters (ϕ_1 and ϕ_3), are hard to estimate by using a monocular camera [184]. The low improvement for ϕ_3 may be due to the DNN's difficulty in recognizing whether a car is within 25m in the same lane as ego. More than 83% of the violations occur in the 6th split where Town 10 is used for testing. This town contains parked cars that can confuse the DNN, causing it to fail to satisfy the property thus decreasing conformance. At the bottom of the graph we note that T4PC was not able to provide gains for ϕ_6 (0.00%) as $\mathcal{B}$ already does not violate the property.

Beyond violation improvements, the mostly central position of the dots and crosses indicate that the main losses for most properties do not vary much from the baseline. The average loss improvement percentage for both losses is -0.27. However, the reduction in violations for ϕ_3 led to more

[12] Since the 2 props and 4 props sets contain 5 and 10 more property combinations than 1 prop, we scale up the number of $\mathcal{B}$ violations (number to the left of the colored arrows) to serve as a baseline.

[13]Since we train a single DNN to satisfy the 6 properties simultaneously, the main loss differences (dots and crosses) are the same for all properties.

noticeable negative loss improvements, likely due to the DNN's inability to identify preconditions, thus biasing predictions towards positive accelerations. The case study in Section 4.2.3 explores this further by deploying such DNNs in simulation.

The trends observed when improving conformance of single properties extend mostly to multiple properties. As we increase the number of properties included in the optimization (2, 4, and 6 props in Figure 4.12), we observe that violation reductions achieved by T4PC for individual properties decrease, while the main losses improve. This trend can be attributed to the increased competition among properties: when multiple constraints are optimized together, their effects on the network's outputs can counterbalance one another. For example, some properties may encourage acceleration while others promote deceleration, or may favor steering in opposite directions. This interplay among property objectives balances the network's behavior, leading to improved performance on the main losses.

We also note that the properties that perform the best have the largest amount of data satisfying their preconditions, have preconditions with easier-to-detect features, or both. For example, ϕ_5 and ϕ_2—which have around 210k and 120k datapoints per split satisfying the preconditions, respectively—exhibit higher violation reductions than other properties. This is not only due to their broader coverage in the dataset, but also because their preconditions are simple to identify: ϕ_5 involves detecting a curve to the right, which implies being the rightmost lane, and ϕ_2 corresponds to the presence of a yellow or red traffic light. Although ϕ_4 has the fewest datapoints satisfying its preconditions (23k), it still performs relatively well, likely because its precondition—detecting a green traffic light—is easy for the network to recognize. In contrast, ϕ_3 has 57k datapoints satisfying its precondition but performs worse, likely due to the greater difficulty of detecting entities within 25 meters, as discussed earlier in this section. This imbalance in both coverage and precondition complexity helps explain which properties benefit most from the optimization.

Despite the competition, we highlight that none of the properties experiences a negative violation improvement, demonstrating that even with multiple properties pulling the model in different directions, the optimization process yields consistent safety benefits without introducing regressions.

T4PC is able to **train** DNNs optimizing for **multiple properties** at the same time, reducing property violations and in 77% of cases improving the main losses.

4.2.2.3 RQ#2: Training-time efficiency with multiple properties

To evaluate the efficiency of T4PC, we consider two dimensions: (1) the training-time overhead incurred when targeting an increasing number of properties simultaneously, and (2) the trade-offs between training time and performance gains as models are trained for longer periods.

Efficiency in Terms of Training Time

We assess how well T4PC scales by measuring the per-batch training cost as the number of properties increases. To do so, during the evaluation of RQ1, we instrumented the training phase to record the time taken for each batch, including the forward pass, the calculation of the original loss and, for T4PC, the property loss, as well as the backpropagation step to update the DNN weights.

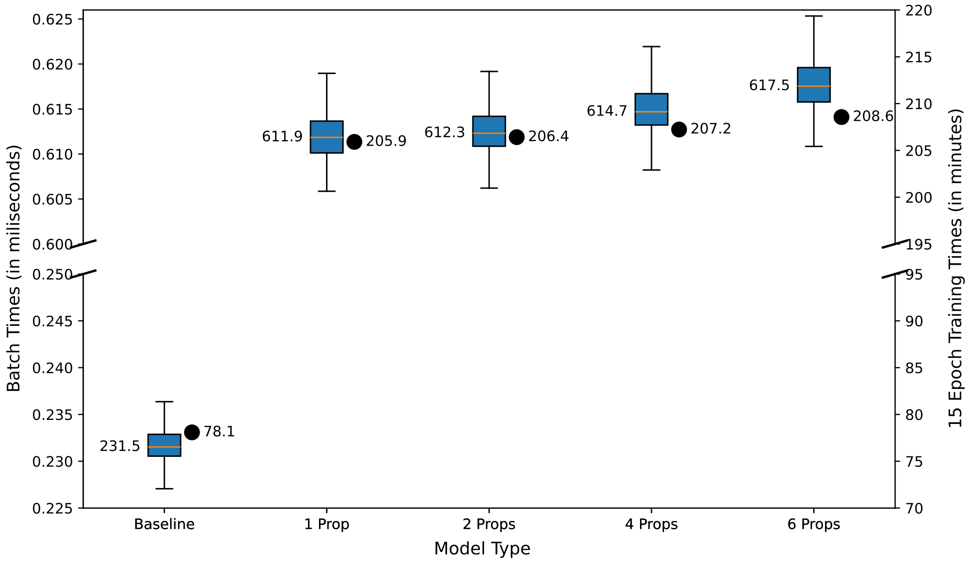


Figure 4.13: Time to train a batch (blue, left-axes) and median time to train for 15 epochs (dot, right-axes), across property sets.

The box plots in blue (left) in Figure 4.13 show the distribution of time taken per batch (measured in milliseconds in the left-axes), while the black dots show the median time taken to train 15 epochs (measured in minutes in the right-axes), all across the number of properties targeted during training. For the baseline models, targeting zero properties, the median batch time is 232ms. When targeting one property, the median batch time is 612ms due to the overhead incurred by T4PC; although substantial in relative terms, the successful application of T4PC through the other RQs demonstrate its feasibility. Increasing to 2, 4, and 6 properties incurs only a minor increase in the median time on the order of a millisecond per additional property, or less than 1%. Through the black dots we can see how these values scale when training the different models for 15 epochs of 1,351 batches each. Neither the box plots nor the black dots include the I/O time of loading images and labels which adds approximately 100 minutes with high variability to both baseline and T4PC. In this context, the overhead of T4PC is practical because it represents a fixed cost rather than a compounding one, ensuring the approach remains scalable. While enabling property checks introduces an initial computational cost, the negligible marginal increase ($< 1\%$) for each additional property ensures that the total training duration remains within the same order of magnitude.

Cost-effectiveness of Training Longer

To examine the cost-effectiveness of training over extended durations, we trained a model with property loss for 72 hours—the maximum time allowed per GPU partition on our SLURM server—using the same hyperparameter and hardware settings described in Section 4.2.2.1 (Experimental Design). We focused on the most challenging setting: optimizing the first six properties from Table 4.6, completing 156 epochs within the time limit. To account for variance, we repeated the experiment three times using the same data split, resulting in three trained models.

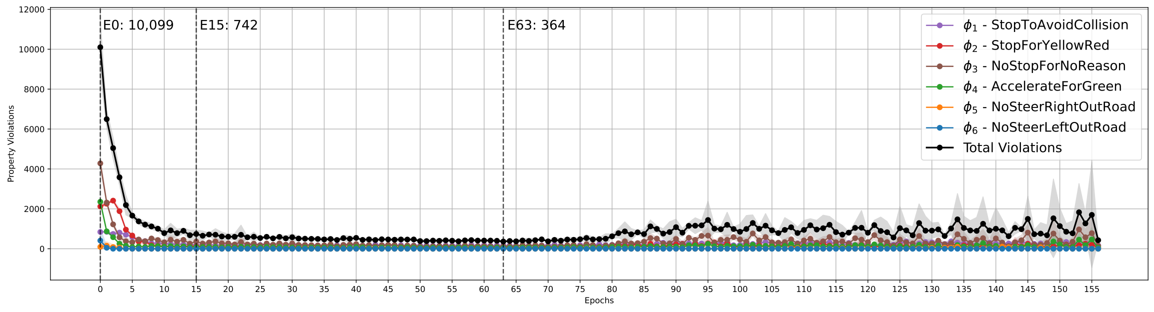


Figure 4.14: Average validation property violations with 2 std interval when training for 72 hs.

Figure 4.14 shows the average total number of property violations computed over the 3 trained models during validation, with a 2 standard deviation interval shaded in gray. At epoch 0, there are 10,099 violations. By epoch 15, the number of violations is 742 (a 92.7% reduction), and by epoch 63, it drops to 364 (a 96.4% reduction)—the minimum average of total violations across all epochs. This additional 3.7 percentage point improvement comes at a significant computational cost: training for 15 epochs takes approximately 5 hours, whereas 63 epochs—required to achieve the additional gain—take 21 hours, representing over 300% more training time.

T4PC **scales efficiently** to the training of multiple properties, adding only minimal overhead per batch, and a significant percentage of the training performance gains are obtained early in the process.

4.2.2.4 RQ#3: Fine-tuning towards property conformance

Setup specific to experiment

To answer this question, we need to fine-tune existing DNNs. We use the 6 baseline DNNs ($\mathcal{B}$) from Section 4.2.2.2 (Setup specific to experiment) as a starting point, fine-tuning each for 15 additional epochs on the same data split and dataset. The fine-tuning process using only the loss function computed from the data labels results in 6 fine-tuned baselines, $\mathcal{B}_{ft}$, while the fine-tuning process using T4PC towards conformance for 6 properties renders 36 new DNNs, $\mathcal{M}_{ft}$. For fine-tuning the DNNs, we use a smaller learning rate of 10^{-5} compared to 10^{-4} when training from scratch, because we want to make small adjustments to the pre-trained weights without losing the model's existing knowledge.

Results

Figure 4.12 with "Ft 1 prop" label summarizes the results. Overall, the violation improvements show a similar trend as in Section 4.2.2.2. All but one property shows clear violation reductions, with the only property not seeing gains (ϕ_6) having the baseline DNN already achieving 0 violations. ϕ_5 achieves a 98.43% violation improvement, and the other 4 properties show between 35.01% and 76.55% violation improvement. For the properties involving acceleration, except for ϕ_1, violation improvement comes at the cost of main loss decreases that are more noticeable than when training the DNN towards property conformance from scratch (shown in Section 4.2.2.2). For example, the DNN acceleration loss for ϕ_4 went from 2.59% when training from scratch to -15.27% when fine-tuning towards conformance. We conjecture that the reduction in main loss arises from the base DNNs trained with data labels having achieved a local minimum with respect to the main loss. Thus, fine-tuning those DNNs for conformance to those properties is likely to have moved them away from that minimum. In such cases, the engineer will have to analyze the effect of increases in main loss in light of the decrease in violations. In Section 4.2.3 we deploy a navigation DNN in simulation to assess the true impact of such losses.

T4PC is able to **fine-tune** existing DNNs, reducing number of violations, albeit with more noticeable differences in main loss, showing its effectiveness at reducing violations of existing highly trained DNNs.

4.2.2.5 Threats to Validity

The validity of our findings across the family of experiments suffers from 3 key threats.

First, the starting navigation DNN, the collected dataset, and the properties selected allow us to conduct a family of studies where we can manipulate the conditions under which T4PC is applied in order to answer multiple research questions. However, the external validity of this selection limits the reach of our findings. The DNN architecture was the same as used in Section 4.1.3, but more complex architectures and parameterization may offer different tradeoffs when training for conformance. The collected dataset also came from CARLA, following the same procedures and environments employed by similar systems [181]. Yet, differences in sensors, their fidelity, and their rates, for example, may impact T4PC. Although the 6 properties we use are representative of many others, there are still many properties that would require a more expressive abstraction and language to be incorporated into T4PC.

Second, the implementation of T4PC involves many components, from the scene graph generator to the Lagrangian optimization, of considerable complexity and with many configurable parameters.

Although we have tested those components in different configurations, they may still contain faults. We make them available, as well as raw and intermediate data, for the community to reuse and assist in improving them at https://github.com/less-lab-uva/T4PC.

Third, from a construct validity perspective, in these experiments the DNNs trained for conformance are judged against a validation set in terms of property violations and main loss. This assessment is limited in that it computes those metrics against an evaluation dataset and the impact of changes in the main loss metric in the absence of an enclosing system context. We address this concern in the next section by applying T4PC to two real autonomous systems, running them in a simulated environment that provides a range of driving metrics.

4.2.3 Case Study on Real Systems

This study aims to broaden the T4PC effectiveness assessment from Section 4.2.2 along 4 dimensions: 1) target two AV systems developed by third parties with different architectures and consuming multiple inputs; 2) fine-tune each system for multiple property conformance; 3) deploy the systems in a series of CARLA simulated environments; and 4) judge the systems using CARLA's driving assessment measures.

4.2.3.1 Setup

Target Systems and Execution Environment

We target the TCP [181] and Interfuser [182] AV systems, two of the three best performing in the CARLA Leaderboard competition[14]. These systems were selected due to their strong performance and architectural diversity. TCP is a fully end-to-end DNN, while Interfuser combines a DNN with a rule-based module written in Python. Since T4PC requires the entire system to be differentiable, we approximate the relevant portions of Interfuser's coded module with a DNN during training to enable the use of our property-based loss function.[15] TCP takes in a single image from an RGB camera, the

[14]The ranking was hosted at `https://paperswithcode.com/sota/autonomous-driving-on-carla-leaderboard`, which is no longer available.

[15]In principle, any coded module could be approximated using a data-driven method such as a DNN. However, such approximations become more difficult and require more data and sophisticated architectures as the module complexity increases or incorporates internal states.

ego's speed, and the target waypoint. Its architecture includes a ResNet34 [166] image encoder, and two GRU [185] branches for steering angle and acceleration predictions. In contrast, Interfuser takes in images from 3 RGB cameras, a LiDAR point cloud, the ego's speed, and the target waypoint. It features a ResNet50 image encoder and a ResNet18-based LiDAR encoder. These features are fused using a transformer encoder-decoder, which outputs ten future waypoints, a semantic feature map, and traffic-related information, including traffic light status, stop sign presence, and intersection detection. These outputs are then passed to a coded controller, which produces the final steering angle and acceleration commands. As we only target acceleration-based properties for Interfuser - explained in Section 4.2.3.1 (Properties) - we only need to approximate the acceleration portion of the controller. To do so, we collected the inputs it consumes and the acceleration outputs it produces over the same dataset, and trained a DNN that achieved 0.04 Mean Absolute Error. Both systems produce a steering angle in the range $[-1, 1]$, representing -70 and 70 degrees, and acceleration in the range $[-1, 0.75]$. We use publicly available pretrained weights from the respective repositories. For evaluation, both AV systems are deployed in simulation on ten routes in Town 05, which are excluded from training and validation. This town includes a variety of roads (e.g., 2-lane roads, 3 and 4-lane highways, and T intersections) and a variety of entities (e.g., traffic lights, stop signs, pedestrians, cyclists, cars, and trucks).

Properties

Despite its high ranking, we observed that TCP struggled to comply with stop signs. To address this, we defined two properties (ϕ_7 and ϕ_8 in Table 4.6) targeting stop sign behavior. ϕ_7 enforces stopping at a stop sign, while ϕ_8 promotes resuming motion once a full stop has occurred. The preconditions of these two properties differ slightly; ϕ_8 includes a constraint requiring no entity to be within 7 meters ahead, ensuring safe acceleration, while ϕ_7 does not. We adopt a 0.1 m/s speed threshold—used by CARLA's low-level controller—to define when the vehicle is considered "stopped." If a stop sign controls the ego lane and speed exceeds this threshold, the vehicle should decelerate; otherwise, it should accelerate.

Interfuser exhibited violations such as running red lights or colliding with pedestrians and other vehicles. These failures align closely with the behaviors targeted by several of the properties used in the controlled experiment. As a result, we fine-tune Interfuser with properties ϕ_1–ϕ_4 from Table 4.6.

Dataset

For training TCP with T4PC, we reused the dataset from the controlled experiments described in Section 4.2.2, which includes single RGB images, ego speed, and target waypoints, along with the labels required by TCP [186]. Unlike the original TCP that used 4 towns for training and 4 for validation, we used 3 towns for training (1, 4, and 10), and 3 for validation (2, 5, and 7) because we collected 6/8 towns as explained in Section 4.2.2.1 (Dataset). For training Interfuser with T4PC, we collected a new dataset using its official data collection scripts in CARLA. The dataset contains **138,585** frames, each with 3 RGB images, a LiDAR point cloud, steering angle, acceleration, and a scene graph. Data was collected across 6 CARLA towns, and we followed Interfuser's standard procedure of executing multiple routes per town, where each route guides the ego vehicle through a sequence of waypoints. Following the original Interfuser, we used 5 towns (1, 2, 4, 7, and 10) for training[16] , excluding two due to collecting data from only 6 of the 8 towns, as discussed in Section 4.2.2.1 (Dataset), and 1 town for validation (5).

TCP & Interfuser Application

We fine-tune the TCP and Interfuser DNNs released by the authors with and without T4PC[17]. To fine-tune the DNNs, we used the same hyperparameters defined by the authors of TCP and Interfuser. We fine-tuned the DNNs for 5, 10, and 15 epochs for TCP and 3, 5, and 7 epochs for Interfuser, corresponding to approximately 10%, 15%, and 20% of their original training schedules of 60 and 35 epochs, respectively. The decision of 10%, 15%, and 20% matches what is shown in Figure 4.14, where we picked 15 epochs to strike a balance between performance and cost which represents 10% of the 155 epochs. The two greater percentages are meant to sample from the region where better results appeared without incurring significant training costs.

After fine-tuning, we evaluate the DNNs using the 10 evaluation routes described in Section 4.2.3.1 (Target Systems and Execution Environment). To account for variation in training, we train 5 DNNs with T4PC and 5 DNNs without T4PC, using the 3 different number of epochs, for a total 30 DNNs per AV system. Likewise, to account for variation during the evaluation of the DNNs in simulation,

[16]We limit data collection to only the default weather configuration as changing weather is not part of any property.
[17]Fine-tuning the DNNs without T4PC allows us to refine it with the same data we collected from CARLA.

we ran each DNN in CARLA 5 times. We then group the results by epochs trained and whether it was trained with T4PC or not, resulting in 250 data points (5 DNNs by 10 routes by 5 runs), and aggregated them by taking the mean and standard deviation. Of the several metrics CARLA calculates, we report the overall driving score and the average infraction scores that include a statistically significant difference for each system[18].

4.2.3.2 Results

Table 4.7 and Table 4.8 provide a summary of the results for TCP and Interfuser respectively, with the rows alternating between optimization with T4PC and the baseline, for the different percentages of training epochs. The bold values are significantly better, determined by a t-test with p-value of 0.05.

Treatment	Driving Score ↑	Collision Vehicles ↓	Stop Sign Infraction ↓	Vehicle Blocked ↓
T4PC 5	81.09±21.87	0.13±0.42	**0.54±0.72**	0.09±0.29
Base 5	79.20±23.06	0.12±0.41	0.98±1.23	**0.00±0.00**
T4PC 10	81.98±21.31	0.16±0.48	**0.59±0.79**	0.03±0.17
Base 10	78.37±22.48	0.11±0.38	1.02±1.17	0.01±0.09
T4PC 15	**81.81±20.47**	0.10±0.35	**0.76±0.95**	0.02±0.13
Base 15	75.99±23.50	0.16±0.42	1.06±1.18	0.01±0.09

Table 4.7: TCP scores and infractions. Values in bold are statistically significantly better.

Treatment	Driving Score ↑	Collision Pedestrians ↓	Red Light Infraction ↓	Route Timeout ↓
T4PC 3	58.40±31.83	0.21±0.44	0.42±0.64	0.20±0.40
Base 3	59.44±31.74	0.19±0.43	0.44±0.61	0.22±0.41
T4PC 5	63.90±31.31	0.16±0.38	**0.24±0.44**	0.17±0.37
Base 5	59.45±30.67	0.15±0.38	0.43±0.64	0.21±0.41
T4PC 7	62.48±33.38	**0.17±0.39**	**0.19±0.40**	0.23±0.42
Base 7	60.15±32.21	0.28±0.51	0.42±0.60	**0.14±0.34**

Table 4.8: Interfuser scores and infractions. Values in bold are statistically significantly better.

Table 4.7 shows that TCP's stop sign infractions, the one that we aim to reduce, decreased

[18]Other metrics did not lead to statistically significant differences.

significantly (bold) with T4PC, by 38% on average. Meanwhile, the driving score column shows that T4PC also increased the driving score in all cases, on average by 5%, with the improvements by the DNN trained with 20% of the epochs deemed significant. The other two columns provide insight into the side-effects of optimizing for ϕ_7 and ϕ_8 on other types of infractions (collisions with vehicles and vehicles blocked). For models trained with 10% of epochs, we find that prioritizing these properties leads to a significant increase in vehicle blocked infractions. However, when training for 20% of epochs, T4PC renders similar results for the vehicle blocked infractions, while reducing infractions from collisions. This suggests that training TCP for more epochs allows it to perform significantly better, while not making other infractions worse. Overall, these results show that the TCP models trained with T4PC using the 2 properties defined above can significantly reduce the number of stop sign infractions and slightly increase the CARLA driving score, a tall order given that TCP is one of the leaders in the CARLA leaderboard.

Table 4.8 shows that Interfuser's red light infractions are significantly reduced with T4PC when training for 15% and 20% of epochs, and pedestrian collisions are significantly reduced when training for 20% of epochs, both properties targeted by T4PC. The driving score improves with models trained with 15% and 20% of epochs using T4PC, though the changes are not significant, and the route timeout infractions are significantly worse for models trained with 20% of epochs. This could be explained by two main factors. First, the properties may not be specific enough, e.g., ϕ_3 is designed to make the ego vehicle move if there is nothing in front and in the same lane but does not consider cases where multiple lanes overlap (intersections), not helping Interfuser move in those situations. Second, the DNN approximation we trained for mimicking the Interfuser's Python component, although 95% accurate, can introduce noise when training the Interfuser DNN. We address this challenge later in Section 5.3 by introducing an approach to correct the AV behavior in a black-box manner, avoiding the need to re-train the DNNs. Further, there may exist latent faults within Interfuser's Python component that are beyond the reach of T4PC's ability to improve the system's performance through training. Although T4PC was less effective overall at significantly decreasing Interfuser's violations as compared to TCP's, these results indicate T4PC's ability to provide benefit for heterogeneous architecture systems as well.

T4PC is able to **fine-tune two existing open-source AV systems** optimized for the CARLA competition, **decreasing the number of infractions**, and improving their overall driving score.

4.2.3.3 Threats to Validity

This case study has similar threats to the previous experiments. Although it reduces the external validity and construct threats by utilizing two third-party DNNs, deploying them in the widely used CARLA simulation environment, and using the environment's builtin metrics to judge performance. Still, the simulation environment has limitations as the findings may not translate to real deployed systems equipped with DNNs, limiting the generalization to real-world applications, as explained in Section 3.1.3.5.

4.2.4 Summary

This section detailed T4PC, an approach for integrating property conformance into ADS training. The method uses a property-based loss function to penalize unsafe model actions. The evaluation demonstrated that this approach reduces property violations in models trained from scratch and fine-tuned. A case study further showed its application on third-party AV systems, successfully reducing their driving infractions. While the properties enforced in this study were selected to align with the single-frame nature of the target models, we explore the monitoring of richer, temporal specifications in Section 5.1.

4.3 Conclusion

This chapter addressed the problem of proactively embedding safety into the ADS development lifecycle by resolving two distinct challenges: the difficulty in quantifying whether training data adequately covers safety-critical scenarios, and the inability of standard training processes to explicitly enforce adherence to safety rules.

The first contribution, S^{3}C, tackles the data adequacy challenge by utilizing scene graphs to create interpretable, semantic abstractions of sensor inputs. This approach enables the automated quantification of dataset coverage, allowing developers to discriminate between semantically distinct scenarios and identify specific gaps in the training data that correspond to rare corner cases.

Building on the assurance of data adequacy, the second contribution, T4PC, addresses the challenge of ensuring the model learns safe driving behaviors, by integrating safe driving properties directly into the DNN's optimization loop through a custom loss function. Collectively, S^{3}C and T4PC constitute the development-phase components of the SD4AS framework, providing the necessary tools to quantify input diversity and enforce output safety prior to system deployment.

Chapter 5

Increasing property conformance in ADS deployment

This chapter addresses the challenge of ensuring ADS conform to safe driving properties during deployment. While pre-deployment validation is important, the unpredictable nature of open-road conditions, compounded by the uncertainty of ML components which can misperceive or hallucinate objects, requires in-field mechanisms to continuously uphold safety standards. To address this challenge, this chapter presents three interconnected contributions of the SD4AS framework that build a solution, from passive monitoring to proactive correction of ADS behavior.

The first contribution is an approach for synthesizing runtime monitors directly from formal property specifications to evaluate them over time. This approach leverages the abstractions developed in Chapter 3 along with SGL + LTL_f specifications to bridge the semantic gap between high-level safety rules and low-level sensor data. It uses SGs as an intermediate representation, allowing it to translate complex sensor inputs into a structured format upon which safety properties can be evaluated. To evaluate the temporal aspects of driving rules, the properties are converted into Deterministic Finite Automata (DFA). These DFA track the state of the ADS over consecutive time steps, enabling the detection of violations over time. This approach, presented in Section 5.1, establishes the feasibility of continuous monitoring within a controlled simulation environment where privileged information from the simulator—such as ground truth object positions and attributes—can be directly accessed to construct accurate SGs. This initial success demonstrates the feasibility of detecting safe driving property violations during the operation of ADS.

While the first contribution demonstrated the feasibility of synthesizing effective runtime monitors, its reliance on privileged information from the simulator to construct SGs limited its applicability to real-world deployments. To transition this capability from simulation to practice, the second contribution addresses the challenge of generating accurate SGs directly from real-world camera images. To achieve this, we fine-tuned a family of Visual Language Models (VLMs) to generate scene graphs of the driving domain. The VLMs utilize input from a single camera, making it broadly applicable to systems with different camera sensors. This advancement enabled the monitoring approach to interpret complex driving scenarios directly from sensor inputs. By reducing the dependency on simulation-based data, this step demonstrated the potential for real-world monitoring.

Finally, the third contribution transitions from passive monitoring to proactive correction of the ADS behavior. This leap to proactive correction required a low-latency, detailed interpretation of the environment. To achieve this, the third contribution first introduces a new SGG. We engineered an SGG using UniAD, a model that provides 3D object detection and lane segmentation, by implementing algorithms to compute spatial and distance relationships from 3D bounding boxes, while accounting for occlusions, and to correlate object positions with road segmentation maps for establishing lane associations. This faster and more detailed SG representation provides the input for the second part of this contribution: a corrector mechanism. This mechanism enforces property conformance by minimally adjusting the ADS's control outputs—such as steering and acceleration—when a violation is imminent. Together, these three contributions provide an approach to monitor and enforce safe driving properties, thereby increasing the safety of ADSs in deployment The following sections will detail the approach and evaluation of each of these contributions.

5.1 Monitoring Safe Driving Properties with Scene Graphs

Chapter 3 presented an expressive language for defining safe driving properties. Building on it, this section introduces Scene Graph Safety Monitoring (SGSM), an approach that automatically synthesizes runtime monitors from safe driving properties. It translates a high-level safe driving property, specified using SGL + LTL$_f$, into a Deterministic Finite Automaton (DFA). The transitions of this DFA are determined by evaluating the SGL propositions against the SG representation of the environment. This section first details the SGSM approach and subsequently presents its evaluation in a controlled simulation, demonstrating its effectiveness in detecting property violations.

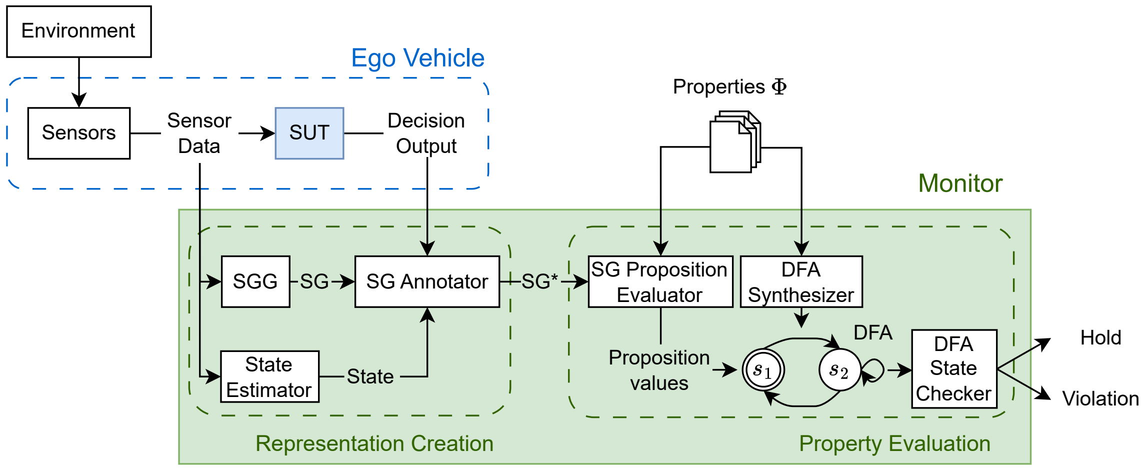
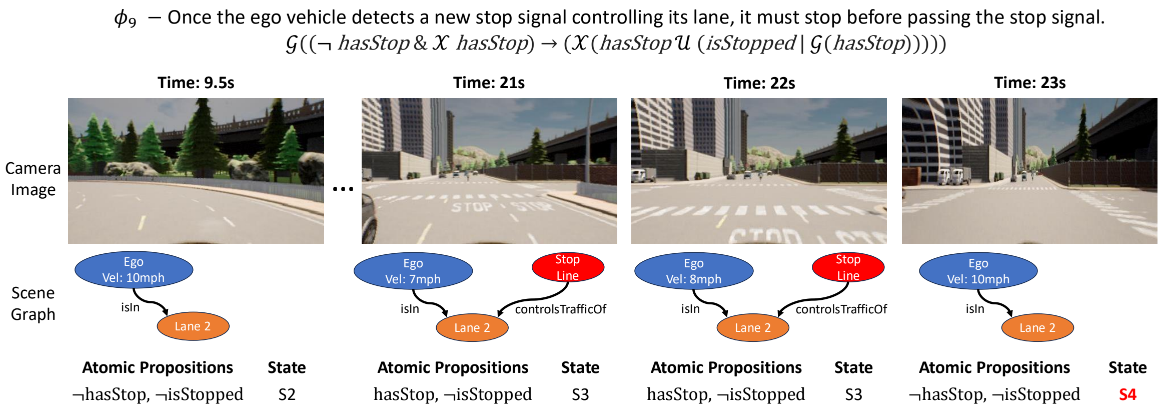


Figure 5.1: SGSM overview.

5.1.1 Approach

As illustrated in Figure 5.1, the ego vehicle perceives the surrounding environment through its sensors, providing sensor data to the System Under Test (SUT). The SUT processes this data to produce a decision output, such as a steering and acceleration command. Concurrently, the SGSM monitor creates a scene representation that is then used to evaluate the set of safe driving properties. To illustrate this process, we will follow the running example of an AV failing to stop at an intersection, where the property *"Once the ego vehicle detects a new stop signal controlling* *its lane, it must stop before passing the stop signal"* is violated. Its definition using SGL + LTL$_f$ is $\mathcal{G}((\neg hasStop \wedge \mathcal{X}(hasStop)) \rightarrow (\mathcal{X}(hasStop\ \mathcal{U}\ (isStopped \vee \mathcal{G}(hasStop)))))$. The property DFA is defined in Figure 5.3 and the violation sequence is shown in Figure 5.2.



ϕ_9 – Once the ego vehicle detects a new stop signal controlling its lane, it must stop before passing the stop signal.
$\mathcal{G}((\neg\ hasStop\ \&\ \mathcal{X}\ hasStop) \rightarrow (\mathcal{X}(hasStop\ \mathcal{U}\ (isStopped\ |\ \mathcal{G}(hasStop)))))$
Time: 9.5s
Time: 21s
Time: 22s
Time: 23s
Camera Image
Scene Graph
Ego Vel: 10mph
Ego Vel: 7mph
Ego Vel: 8mph
Ego Vel: 10mph
isIn
Lane 2
Stop Line
controlsTrafficOf
Atomic Propositions
State
¬hasStop, ¬isStopped
S2
hasStop, ¬isStopped
S3
hasStop, ¬isStopped
S3
¬hasStop, ¬isStopped
S4

Figure 5.2: AV (TCP) running an intersection without stopping. Top: Property defined in english and SGL + LTL$_f$, and camera images. Middle: SG for checking safety property ϕ_9 in Table 3.4. Bottom: Atomic Propositions evaluated from SG* and updated state of the DFA shown in Figure 5.3 leading to violation.

5.1.1.1 Representation creation

This module is responsible for constructing a comprehensive representation of the driving scene at each timestep. This process begins with two components that operate in parallel on the raw sensor data. First, the SGG component consumes the sensor data to produce a SG, which captures the entities and their relationships in the environment. The specific information captured by the SG is driven by the property specifications being monitored. For instance, if the property evaluates correct behavior at a stop sign, the SGG must be configured to extract stop signs and their relationships with the ego vehicle. The progression in Figure 5.2 illustrates this: at time 9.5s, the SGG generates a SG with nodes for the `Ego` vehicle and `Lane 2`, connected by an `isIn` relation. Later, at 21s, as the stop line becomes visible, the SGG updates the SG to include a `Stop Line` node that `controlsTrafficOf` the lane ego vehicle is in (`Lane 2`). Second, the State Estimator component processes the sensor data to determine the internal state of the ego vehicle. This state includes critical information such as its longitudinal or lateral speed, or any other relevant state information, e.g., the blinker status.

Lastly, the SG Annotator component adds information from different sources into the SG. It takes the SG from the SGG, the ego vehicle's state from the State Estimator, and the decision output from the SUT. The result is an enriched scene graph, denoted as SG*, which provides a holistic snapshot of the relevant environment, the ego vehicle's state, and its intended action. This annotation is shown in Figure 5.2, where the state estimation component determines the ego vehicle's velocity at each step, and the SG annotator incorporates it in the SG* as an attribute of the `Ego` node, such as `Vel:10mph` at 9.5s.

5.1.1.2 Property evaluation

The second module uses the enriched SG* to evaluate a given safe driving property, specified using SGL + LTL$_f$, like ϕ_9 from the running example in Figure 5.2. The evaluation of this property begins in the SG Proposition Evaluator component. This component reads the preconditions of the property, which are defined as Atomic Propositions (APs) in SGL, and queries SG* to determine whether each proposition is true or false at the current timestep. The evaluation of APs over the SG yields an understanding of the spatial and temporal distribution of entities related to ego. This process is detailed in the bottom row of Figure 5.2. At 9.5s, for instance, the query for `hasStop` is false because there is no stop sign controlling the traffic of the lane ego is in. Furthermore, the query for `isStopped` is also false because the speed is 10mph. Later, at 21s, the stop line is present, so `hasStop` becomes true, but since the vehicle is still moving, `isStopped` remains false.

LTL$_f$ Formula for ϕ_9: $\mathcal{G}((\neg\, hasStop\ \&\ \mathcal{X}\ hasStop) \rightarrow (\mathcal{X}(hasStop\ \mathcal{U}\ (isStopped \mid \mathcal{G}(hasStop)))))$

Atomic Propositions:
hasStop: | *relSet*(*Ego*, isIn) ∩ *relSet*(*stopLine*, controlsTrafficOf) | > 0
isStopped: | *filterByAttr*(*Ego*, speed, λ x: x < ε) | = 1

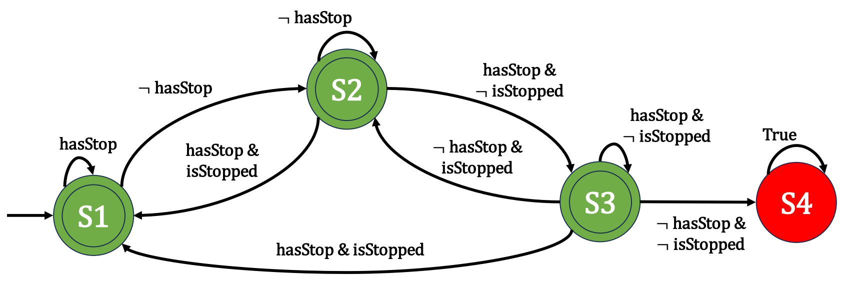


Figure 5.3: LTL$_f$ for safe driving property, Atomic propositions, and DFA for property ϕ_9 in Table 3.4

The boolean proposition values are used to update the state of a Deterministic Finite Automaton (DFA). The DFA is automatically compiled from the LTL$_f$ specification by the DFA Synthesizer component. This synthesis occurs as an initialization step, where the synthesizer takes the LTL$_f$ definition of a property and generates the corresponding DFA used for runtime checking. The synthesizer follows the technique described in [187] and implemented in LTLf2DFA [99] to convert the LTL$_f$ into a DFA. For the running example ϕ_9, the synthesizer processes its LTL$_f$ formula and produces the DFA shown in Figure 5.3.

During runtime, the last monitor component, DFA State Checker, examines the DFA's new state at each timestep to determine the property's status. If the DFA is in a violating state, the monitor outputs a violation signal; otherwise, the property is considered to hold. The effect of evaluating these propositions on the DFA from Figure 5.3 is illustrated by the state changes in Figure 5.2. Initially, the DFA is in state *S2*. At 21s, the propositions (`hasStop` $\wedge$ ¬`isStopped`) cause a transition to state *S3*. The DFA remains in *S3* at 22s because the evaluated propositions are unchanged. Finally, at 23s, the vehicle passes the intersection without stopping, making `hasStop` false and causing a transition from *S3* to the violation state *S4*.

5.1.1.3 Limitations

SGSM has two main limitations. First, the current implementation relies on a simulation-based SGG that leverages privileged ground-truth information from the CARLA simulator. While this enables the creation of accurate SGs to validate the monitoring logic, it does not simulate the uncertainty or noise inherent in real-world perception systems. The approach assumes that the SG is a faithful representation of the environment; consequently, in a real-world deployment, any perception errors or omissions would propagate directly to the property evaluation module, leading to incorrect monitor verdicts. To address this limitation and transition to real-world application, the following section explores generating SGs directly from camera images using VLMs.

Second, manually encoding safe driving rules using the SGL and LTL_f format imposes a burden on the developer. Translating natural language driving manuals into precise formal logic requires expertise in both the domain regulations and the formal specification language. This manual process is time-consuming, subject to errors, and increases the difficulty of covering a broad variety of rules from driving manual of different places.

5.1.2 Evaluation

To assess SGSM effectiveness in monitoring safe driving properties for autonomous vehicles (AVs), we set the following research question:

- RQ: To what extent can SGSM automatically detect safety property violations across a range of contemporary autonomous driving systems?

To answer this question, we first outline the experimental setup within the CARLA simulator. We then present the results of applying SGSM to three different AV systems, providing insights into the framework's strengths and limitations in real-time safety monitoring.

5.1.2.1 Setup

To evaluate SGSM's ability to act as an automatic safety monitor, we need a common execution environment on which to run several AV systems to monitor.

Common Execution Platform

For running the study, we used the CARLA simulator and the 3 top-ranked systems of the CARLA Autonomous Driving Leaderboard as of June 2022[1], using the provided evaluation routes for Town05 described in Section 4.2.2.1 (Target Systems and Execution Environment). The CarlaSGG, detailed in Section 3.1.3, interfaces with CARLA to extract ground truth SGs. We use the default entity and relationship scheme from prior work on SGs for AVs [70, 30], enriched with additional information about which roads and junctions lanes belong to. While our simulation-based SGG uses ground truth information to eliminate the effects of sensor noise in our study, the current trajectory of SGG research in conjunction with the availability of HD maps for AV systems is promising for implementation of SGSM on real-world systems. The following sections explore SGG designed to enable monitoring without relying on simulation ground truth.

Properties

We evaluate the set of nine properties introduced in Table 3.4. We derived these properties from the Virginia Driving Code [161] to cover several categories of driving behavior. There are rules for lane discipline, such as staying on the road (ϕ_2, ϕ_3), avoiding the opposing lane (ϕ_1), and limiting the time spent straddling lanes (ϕ_7). Other properties address vehicle interaction and progress, specifying minimum following distances (ϕ_4), appropriate throttle control when approaching vehicles (ϕ_5), and avoiding unnecessary stops (ϕ_6). The set also includes properties to adhere to traffic controls, such as the requirement to stop for stop signals (ϕ_9), and efficient junction traversal (ϕ_8).

As ϕ_4, ϕ_7, and ϕ_8 contain a threshold parameter, we instantiate 3 versions of each, for a total of 15 property instances leading to 15 synthesized monitors. For ϕ_4, we chose $S \in \{5, 10, 15\}\frac{m}{s}$ to represent parking-lot, urban, and suburban driving speeds. For ϕ_7, empirical studies have found that lane changes take $4.6s$ on average with a standard deviation of $2.3s$ and a maximum of $13.3s$ [188]; thus we select $T \in \{5, 10, 15\}s$ to represent the average, 2 standard deviation, and beyond the maximum time. For ϕ_8, we select $T \in \{5, 10, 15\}s$ as the time to clear the intersection as a left turn across a 4 lane road at $10mph$ takes $5s$, and we allow for a buffer factor of $1 - 3\times$.

AV Systems Evaluated

[1]The ranking was hosted at `https://paperswithcode.com/sota/autonomous-driving-on-carla-leaderboard`, which is no longer available.

Each studied AV system takes in a list of waypoints from the intended route and produces at each frame a control for steering, throttle, and brake; each system has different sensors and software stack. *Interfuser [182]* consists of a DNN with a transformer [189] architecture that feeds a controller that generates a set of actions. It takes 3 images from 3 RGB cameras and a cropped version of the center image to focus on distant traffic lights, a LiDAR point cloud, and the GPS coordinates of the goal location and computes a set of waypoints, an object density map, traffic light state, stop sign presence, and if the vehicle is in a junction. These are fed into the controller to produce the output. *TCP [181]* takes in 1 image from an RGB camera, ego's speed, and the GPS coordinates of the goal location and uses a DNN composed of a CNN-based image encoder using ResNet34 [190], and two GRU [191] branches for trajectory and control predictions. *LAV [192]* consists of a perception DNN, motion planner, and controller. The DNN consumes 3 images from 3 RGB cameras and a LiDAR point cloud, and outputs a BEV map which is fed to the planner along with the next waypoint coordinates to produce the next 10 future waypoints. The waypoints are passed to the controller along with a braking signal from a binary DNN classifier to compute the output.

5.1.2.2 RQ: AV Safety violations

We ran each AV system through 10 routes in Town05 and evaluated all 15 properties at a rate of $2Hz$. This sampling rate was selected based on the performance of the SGSM components. The SGG and annotator required an average of 288 ms to create an SG^*, and the evaluation of all properties required an average of 67 ms. This resulted in a total average processing time of 355 ms. We, therefore, selected a $2Hz$ rate, as its $500ms$ cycle provided a sufficient conservative margin for the 355 ms average processing time to ensure reliable operation. The properties we monitor in this study, such as traversing an intersection or changing lanes, occur over several seconds. Consequently, a 2Hz rate is sufficient to monitor these behaviors and validate whether the AV systems conform to the properties.

Table 5.1 shows how many of the 10 routes resulted in a violation for each AV system for each of the 15 instances of the 9 properties. Note that since the properties are defined as global properties, we track only the first violation, for a maximum of 10 possible violations per AV system per property.

Property System	ϕ_1	ϕ_2	ϕ_3	ϕ_4 S=5	 S=10	 S=15	ϕ_5	ϕ_6	ϕ_7 T=5	 T=10	 T=15	ϕ_8 T=5	 T=10	 T=15	ϕ_9	Total
InterFuser	3	0	10	0	0	0	3	9	10	5	5	10	5	5	7	72
TCP	6	0	10	0	0	0	2	6	5	3	3	8	0	0	8	51
LAV	6	1	10	0	0	0	3	2	8	6	5	10	6	1	7	65
Total	15	1	30	0	0	0	8	17	23	14	13	28	11	6	22	188

Table 5.1: Count of Routes (10 total) with Property Violations by AV System.

We find that the total number of violations ranges from 51 for TCP to 72 for Interfuser (over 150 possible violations).

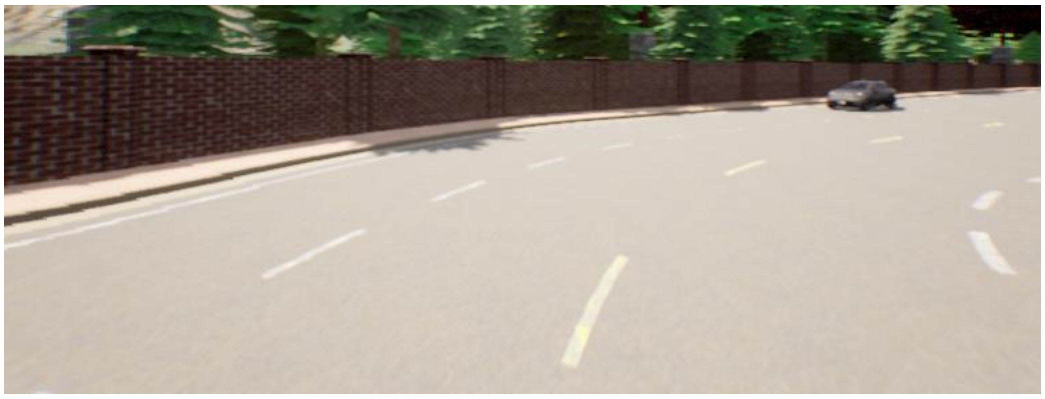


Figure 5.4: Interfuser AV violating ϕ_1 ("Ego vehicle cannot be in the opposing lane"). The vehicle failed to navigate the road curve and crossed into the opposing traffic lane.

We now exemplify the violations found. Figure 5.4 shows an instance of Interfuser violating ϕ_1; as the road curved to the right, Interfuser did not steer right enough and drifted into the opposing lane. Figure 5.5 shows LAV violating ϕ_2, turning left too sharply and driving onto the median between two lanes. While this is not off of the road bed, the SGG denotes it as off road because it is not part of a defined lane of traffic. Fig. 5.2 shows TCP violating ϕ_9 over a series of frames. TCP approaches a junction with a marked stop line, but it does not stop and enters the junction.

Property violation statistics also give information on the driving style of the AVs. None of the AVs violated ϕ_4, which means that they maintained a sufficient distance from the lead vehicles. However, Interfuser and TCP violated ϕ_6 over more than half the routes, i.e., they stopped in the middle of the roadway. Upon further inspection we found that there are 9 cases where this stoppage

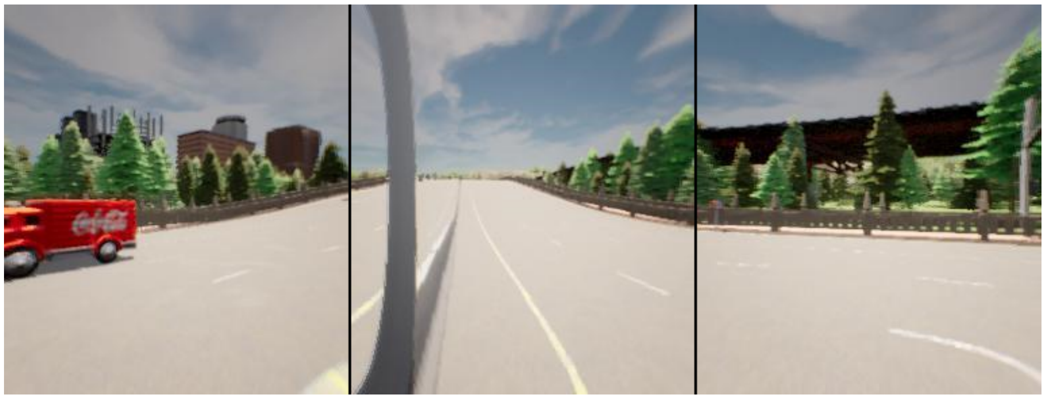


Figure 5.5: LAV AV violating ϕ_2 ("Ego vehicle cannot be out of the road"). After a left turn, the vehicle missed the lane entrance and drove onto the median, which is not considered a lane.

is unjustifiable, but in 4 other cases, the AV is stopping due to a stopped vehicle ahead of it but farther than the 7 meters prescribed in ϕ_6, and in the remaining 4 cases there is a traffic light that is transitioning out of red. This highlights the difficulty in concretizing the parameters used in the specification given the imprecise definitions in the driving manual; 7 meters may be acceptable depending on circumstances. This is further shown in the performance across the parameterizations of ϕ_7 and ϕ_8. As T increases, the specification is more relaxed which leads to fewer violations; e.g. TCP violations are reduced from 8 to 0 under ϕ_8 when T is increased from 5 to 10. However, Interfuser and LAV do not improve as rapidly. This may point to different AV's optimizations; they likely did not accounted for junction crossing times, and instead may have prioritized moving cautiously through a junction leading to slower transits.

5.1.2.3 Threats to validity

In this work we showed the feasibilty of implementing an SG-based monitor and its utility for checking safety property specifications based on driving rules. The external validity of our results, however, is bounded by our use of simulation to create the SGs using ground truth data. Working in simulation enabled us to construct an SGG module that generates accurate SG representations of the world to judge the cost-effectiveness of the framework as a whole, but we recognize that it will be necessary to consider SGGs using various sensor types and in the wild. Moreover, CARLA suffers from the simulation-reality gap [160], so deploying the approach in the real world will be necessary to assess its generalizability. Similarly, more complex operational domain properties should be specified and checked. The internal validity of our results are related to our implementation of SGSM, and to overcome this threat an artifact is available at https://github.com/less-lab-uva/SGSM.

5.1.3 Summary

This section presented SGSM, the first contribution toward addressing the challenge of checking safe driving properties during deployment. The approach translates properties, specified in LTL$_f$, into executable runtime monitors, by synthesizing DFAs and using SGs as the intermediate representation to check the properties. The evaluation, conducted within CARLA, confirmed that SGSM can automatically detect a range of safe driving property violations, such as running stop signs or driving off-road, across multiple AV systems.

While this work demonstrated the effectiveness of SGSM, its implementation relied on privileged, ground-truth information from the simulator to construct the SGs. This limitation, discussed in Section 5.1.1.3 prevents its direct application in real-world scenarios where such perfect information is unavailable. This gap leads to the next challenge, addressed in the following section, which focuses on generating accurate SGs directly from sensor data.

5.2 Visual Language Models as Scene Graph Generators

The SGSM approach introduced in Section 5.1, while effective in simulation, relied on privileged simulator data to construct the SGs necessary for property evaluation. This dependency on ground-truth data presents a barrier to real-world deployment, creating the need to generate SGs directly from on-board sensor inputs, such as camera images. To bridge this gap between simulation and reality, this section investigates the potential of VLMs to accomplish this task serving as scene graph generators. We hypothesize these models are well-suited for this purpose because they have been shown to process complex visual scenes from camera images, possess a broad understanding of objects and relationships from large-scale pre-training, and are improving rapidly due to widespread research and industry attention[2]. However, a systematic evaluation of their effectiveness for this task first requires a suitable benchmark dataset containing real-world images paired with ground-truth SGs. To provide this resource, this section details the development of the DriST dataset. Using this new dataset, the section then presents a systematic evaluation of four distinct querying strategies designed to assess the capability of VLMs to extract SGs from real-world driving scenes. This evaluation provides insights into the most effective methods for extracting SGs from raw visual data using VLMs.

5.2.1 Approach

To systematically investigate the use of VLMs as real-world SGGs, the approach is built upon two primary components. First, we developed a dataset of images paired with ground-truth SG necessary for both benchmarking different VLMs performance on this task and fine-tune them. Section 5.2.1.1 details the process to create the DriST dataset from existing driving datasets. Then, Section 5.2.1.2, describes the inference pipeline to use VLMs as SGGs, which operates by prompting the models through several distinct querying modes. Each mode alters the information provided to the VLM—from general scene-level requests to specific, bounding-box-guided questions—thereby eliciting different outputs.

5.2.1.1 DriST (Driving Scenes with Triplets) Dataset

This section details the DriST (Driving Scenes with Triplets) process to create a large-scale dataset, as illustrated in Figure 5.6. The process leverages existing autonomous driving datasets—specifically NuScenes, Waymo, and KITTI—to generate images annotated with ground-truth SG triplets. It involves standardizing 3D annotations into a common coordinate frame, filtering for occluded objects, and extracting SG triplets based on each object's position relative to the ego-vehicle. The final output is the DriST dataset, which pairs each image with its corresponding SG triplets and bounding boxes. The following paragraphs describe the SG triplet format and the details of this generation procedure.

[2]As of October 2025, 284 VLMs and 31 benchmarks have been released in the past 2 years based on https://huggingface.co/spaces/opencompass/open_vlm_leaderboard

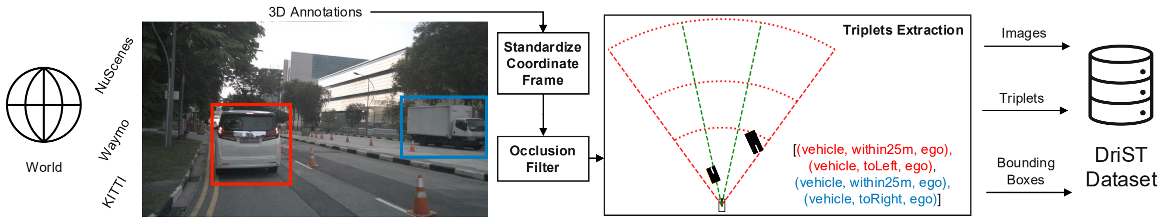


Figure 5.6: DriST dataset creation process.

Scene Graph Triplets

As established in Section 3.1.2, a scene graph is a set of triplets. This work adopts this representation because the triplet structure aligns well with the text generation output of VLMs. Specifically, a VLM can be prompted to describe a visual relationship, and its natural language response can be directly structured as a textual triplet. A triplet of the form <*subject, relation, ego*> captures a single spatial relationship between an object and the ego vehicle, providing a precise description of how the two entities are positioned relative to each other.

The list of potential subjects and relations has a long-tail distribution with many rare instances. In this work, we focus on the head of that distribution. That is, we prioritize entities and relations frequently found in existing datasets and related to safety driving requirements. More specifically, we focus on three types of **subjects**: *vehicles*, *persons*, and *bicycles*. This scope also helps us to maintain compatibility across datasets; for instance, while nuScenes include a wider variety of entity types, Waymo only provides labels for the three subjects mentioned above.

We also focus on six **relations** that capture spatial interactions with the ego vehicle, organized into two categories: relative position and relative distance.

- Relative position describes the lateral positioning of entities within the ego vehicle's field of view, which is divided into three equal regions—*left of*, *in front of*, and *right of* the ego.
- Relative distance captures the proximity of entities to the ego vehicle, categorized into three ranges: *within 25 meters*, *between 25 and 40 meters*, and *between 40 and 60 meters*. These ranges correspond to safe braking distances when traveling at speeds of 25, 35, and 45 miles per hour [178], respectively, reflecting safety considerations.

Extracting Triplets

Autonomous driving datasets such as nuScenes [174], Waymo [193], and KITTI [194] contain sensor data (images, LiDAR) accompanied by 3D annotations for the entities in each scene. However, none of them include annotations of spatial relationships between entities and the ego vehicle. As depicted in Figure 5.6, to generate triplets from the 3D annotations in each dataset, we developed a process that standardizes each scene in a common coordinate frame, performs necessary occlusion filtering, extracts ground truth scene graphs as list of triplets, and associates a 2D bounding box to each subject, to localize it in the image.

Standardize Coordinate System. Each dataset uses a unique coordinate system for 3D bounding boxes, often based on sensor placement or proprietary conventions. To unify these diverse annotations, we convert all 3D bounding boxes to a common North-East-Up (NEU) coordinate system based on the right-hand rule. The origin of this coordinate system is set at the ego vehicle's center, allowing a consistent spatial reference across all scenes.

Occlusion Filter. Spatial reasoning in autonomous driving camera scenes must distinguish visible objects. To determine whether an entity is visible we use ray tracing from the ego vehicle to each entity in the scene. If a ray intersects with any other entity's bounding box, we mark the target entity as occluded and remove it from the list of entities in the scene.

Triplets Extraction. Using the field of view of the camera and a distance threshold of 60 meters (red cone in Figure 5.6), we prune entities whose centroids fall outside this area. We then use the sections of the cone, parameterized based on the relations definitions, to determine the distance and position of the entity relative to ego. For example, in Figure 5.6, there is a vehicle (white van) on the left and within 25m, as we can see that it is in the left bottom sub-area of the red cone. There is also another vehicle (white truck) on the right and within 25m as well, as it is depicted in the right bottom sub-area.

Finally, we transform the 3D bounding boxes of each entity in the scene into 2D bounding boxes, representing each data point as an image, a list of triplets, and these 2D boxes. The bounding box provides two advantages: first, it enables spatial grounding of the triplets. This means we can visually link the textual triplet to a specific, outlined region in the image, allowing for clear visual validation of each entity's position and relationship. Second, it enables more specific query methods. For example, the bounding box can be visually rendered on the image to ask a VLM a question about the specific entity contained within it. This information is used in two of the query modes outlined in the next section.

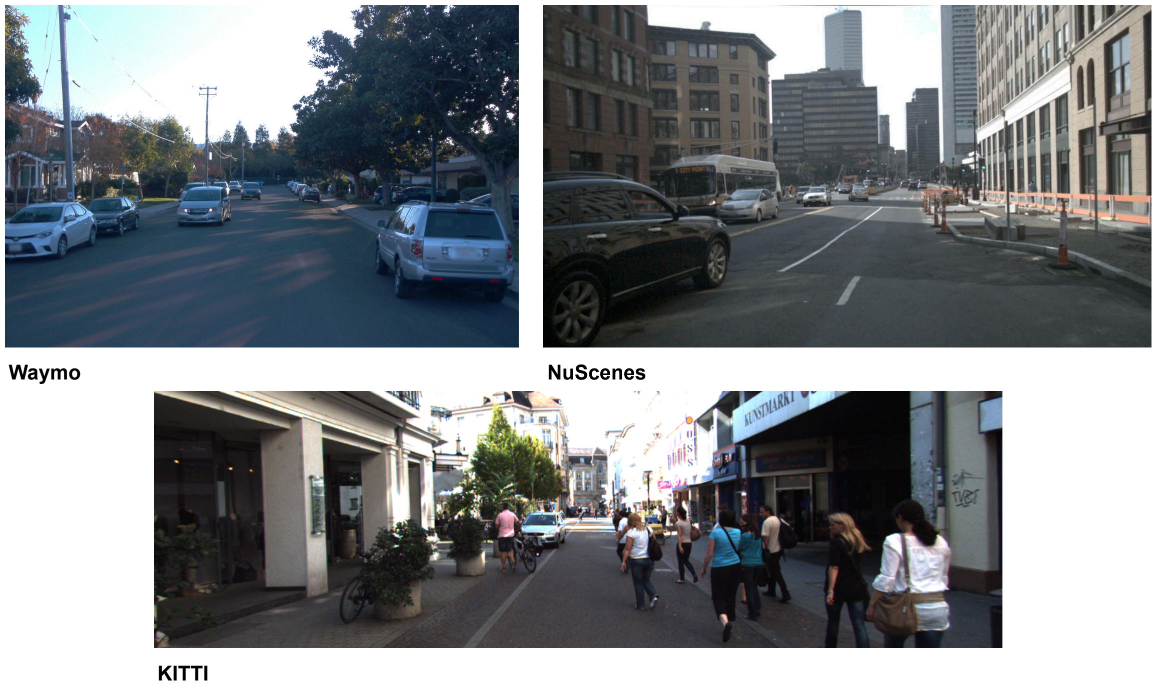


Figure 5.7: Sample images from Waymo, nuScenes, and KITTI.

Dataset

We built the DriST dataset by incorporating the data of three established autonomous driving datasets: nuScenes [174], Waymo [193], and KITTI [194] (samples are shown in Figure 5.7). As detailed in Table 5.2, the combined dataset contains a total of **209,590** images labeled with triplets. On average, each scene contains over 4 entities (2.9 vehicles, 1.6 persons, 0.06 bicycles) and 8 triplets. The most common relationships for position is in-front and for distance is between 40-60m, but all relationships appear on average of at least 1.2 times per image. This frequency is driven by the high object density in many scenes; for instance, the Waymo sample in Figure 5.7 captures 5 cars, while the KITTI sample includes 9 pedestrians. The distribution of entities and relationships, combined with the use of these three source datasets, provides a varied set of annotated driving scenarios. The diversity comes from the different sensor configurations, camera resolutions, and typical driving elements present in the original datasets, which together offer a range of realistic spatial contexts to study the potential of VLMs to reason in this domain.

Dataset	**Labeled Data**	**Train (75%)**	**Val (5%)**	**Test (20%)**
nuScenes	34149	25612	1707	6830
Waymo	167960	125970	8398	33592
KITTI	7481	5611	374	1496
DriST	209590	157193	10479	41918

Table 5.2: Dataset Splits for nuScenes, Waymo, KITTI and DriST.

5.2.1.2 Inference pipeline

The inference pipeline for using VLMs as SGGs is depicted in Figure 5.8. We utilized the concept of a "Query Mode" to define the interaction strategy; we distinguish this from a simple prompt because a single mode generates one or multiple queries (prompts) to extract specific information from the scene. The process begins with the Query Generator, which implements the selected mode. It takes the input image, optional bounding boxes, and a base prompt to generate the natural language queries. This base prompt serves as a template with placeholders (an example for Query Mode 2 is shown in the bottom-left box of Figure 5.8) that the generator populates to create specific questions, such as ""Is there a vehicle within 25 meters?". These queries are then passed to the VLM, which provides a response, such as structured text (e.g., "Yes. 2") or a triplet, depending on the query mode. Finally, the Response Aggregator parses these outputs. In the example shown, the structured text "Yes. 2" would cause the aggregator to add two identical <vehicle, within25m, ego> triplets to the final list of triplets, corresponding to the two vehicles detected within 25 meters. If the VLM output is a triplet, the aggregator simply appends it to the final list of triplets. This process produces a list of structured SG Triplets. To determine the most effective prompting strategy, we designed four distinct query modes, representing progressively granular methods for interrogating the model about the scene's spatial relationships.

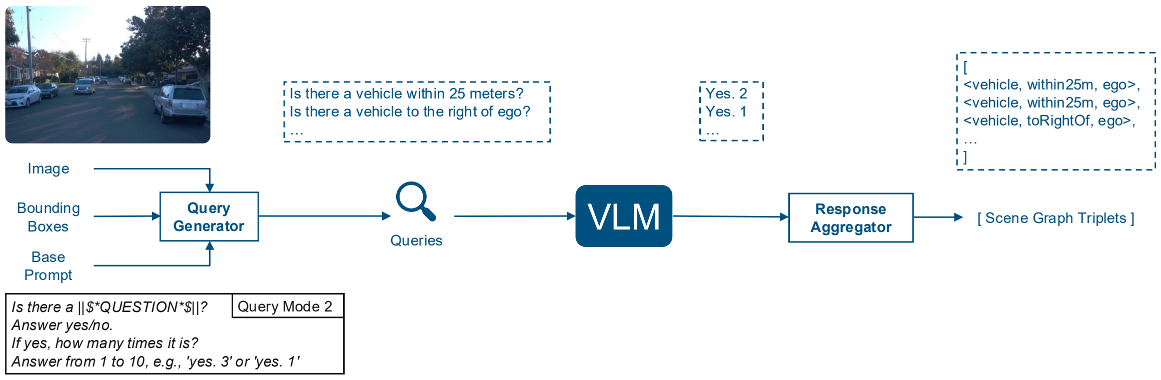

Is there a vehicle within 25 meters?
Is there a vehicle to the right of ego?
…
Yes. 2
Yes. 1
…
[
<vehicle, within25m, ego>,
<vehicle, within25m, ego>,
<vehicle, toRightOf, ego>,
…
]
Image
Bounding Boxes
Base Prompt
Query Generator
Queries
VLM
Response Aggregator
[Scene Graph Triplets]
Is there a ||$*QUESTION*$||?
Answer yes/no.
If yes, how many times it is?
Answer from 1 to 10, e.g., 'yes. 3' or 'yes. 1'
Query Mode 2

Figure 5.8: Scene Graph Generation using a VLM with query mode 2.

Mode 1

This mode involves prompting the VLM with a general request for all spatial triplets in an image. We provide context regarding the entities and relationships of interest and specify that relationships should be expressed relative to the ego vehicle. The base prompt of query mode 1 can be seen in Figure 5.9. The VLM then outputs a variable-length list of triplets corresponding to the entities and their relationships identified within the image. This open-ended querying mode allows us to capture the model's overall capacity to enumerate triplets in a driving scene, by querying the VLM only once. An example of this query mode interaction can be seen in Figure 5.11a.

Mode 2

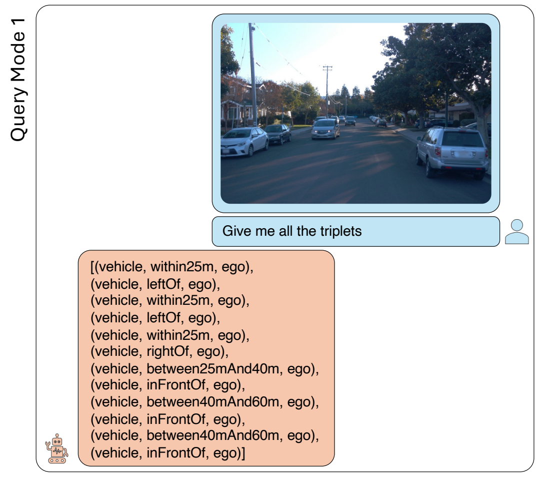
This mode employs a more targeted approach, querying the VLM with yes/no questions for each possible combination of entity type and spatial relationship, e.g., the entity "car" can have one of the three relative positions (left, in front, or right) and one of the three relative distances (within 25 m, between 25 and 40 m, or between 40 and 60 m). The base prompt of query mode 2 can be seen in Figure 5.10 and an example of this query mode interaction can be seen in Figure 5.11b. Following the example, we ask the VLM if there is a car in within 25 m of the ego vehicle, and how many. Each affirmative response like "Yes. 2." is used to generate triplets by the Response Aggregator. When the VLM indicates that there are multiple instances of the entity (2 in this case), the corresponding triplet is repeated accordingly, and added to the list of predicted triplets. This mode constrains the VLMs to perform a more detailed analysis of individual entity relationship pairs.

```
### Definitions:
1.  **ego**:  The vehicle holding the camera and capturing the scene (not visible in the
image).
2.  **OBJECT**:  Refers to one of the following:
- Person
- Bicycle
- Car
3.  **RELATIONSHIP**:  Refers to one of the following:
- Positional:  ‘in_front_of‘, ‘to_left_of‘, ‘to_right_of‘
- Distance-based:  ‘within_25m‘ (within 25 meters), ‘between_25m_and_40m‘ (between 25 and 40
meters), ‘between_40m_and_60m‘ (between 40 and 60 meters)

### Instructions:
1.  Identify all OBJECTS in the scene.  Only include objects that are clearly visible,
identifiable, and relevant in the image.  Do not include objects that are not present.
2.  For each OBJECT, generate two RELATIONSHIPS with the ego:
- One positional RELATIONSHIP (‘in_front_of‘, ‘to_left_of‘, ‘to_right_of‘)
- One distance-based RELATIONSHIP (‘within_25m‘, ‘between_25m_and_40m‘, ‘between_40m_and_60m‘)
3.  Use the following structured format for your output.
4.  Verify that each OBJECT is clearly visible and identifiable in the image before including
it in the output.

### Output Template:
‘‘‘
[
(OBJECT, RELATIONSHIP, ego),
(OBJECT, RELATIONSHIP, ego),
...
]
‘‘‘

### Example:
Given an image with one bicycle and one car, the output should be:
‘‘‘
[
(bicycle, in_front_of, ego),
(bicycle, between_25m_and_40m, ego),
(car, to_left_of, ego),
(car, within_25m, ego)
]
‘‘‘

### Important:
- Do not include any objects that are not visible in the scene.
- Ensure that all identified objects are clearly distinguishable and relevant.
- Verify the presence and relevance of each object before including it in the output.

---

### Image Analysis:
Using the above prompt format, please analyze the following image and generate the output
according to the template, ensuring only visible, identifiable, and relevant objects are
included:
||*IMAGE*||
```

Figure 5.9: Query mode 1 base template.

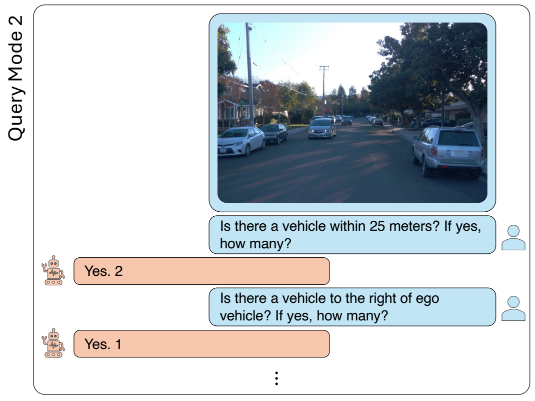
```
||*IMAGE*||
Is there a ||*QUESTION*||?
Answer yes/no.
If yes, how many times it is?
Answer from 1 to 10, for example:  'Yes.  3' or 'Yes.  1'
```

Figure 5.10: Query mode 2 base template.



```
[(vehicle, within25m, ego),
(vehicle, leftOf, ego),
(vehicle, within25m, ego),
(vehicle, leftOf, ego),
(vehicle, within25m, ego),
(vehicle, rightOf, ego),
(vehicle, between25mAnd40m, ego),
(vehicle, inFrontOf, ego),
(vehicle, between40mAnd60m, ego),
(vehicle, inFrontOf, ego),
(vehicle, between40mAnd60m, ego),
(vehicle, inFrontOf, ego)]
```

(a) Query mode 1



Is there a vehicle within 25 meters? If yes, how many?

Yes. 2

Is there a vehicle to the right of ego vehicle? If yes, how many?

Yes. 1

⋮

(b) Query mode 2

Figure 5.11: Examples of Query mode 1 and 2.

Mode 3

This mode leverages bounding boxes to guide the VLM's attention to specific entities within the image. This mode operates by iterating through each entity; for each one, a separate query is made that highlights only that entity's bounding box. We then prompt the VLM to identify the triplets describing that entity's spatial relationships to the ego vehicle. Figure 5.12 shows the base prompt for this query mode, and Figure 5.14a is an example of this query mode interaction with two queries for two different entities using their bounding box. The VLM answers with all the triplets of that entity, that get appended by the Response Aggregator to return the final list of SG triplets. This query mode assumes access to bounding box information. In our experiments, we use ground-truth bounding boxes from the dataset. This allows us to benchmark the VLM's reasoning capabilities in isolation, without errors from a detector. In a practical deployment, this information would come

```
### Definitions:
1. **ego**: The vehicle holding the camera and capturing the scene (not visible in the
image).
2. **OBJECT**: Refers to one of the following:
- Person
- Bicycle
- Car
3. **RELATIONSHIP**: Refers to one of the following:
- Positional: ‘in_front_of‘, ‘to_left_of‘, ‘to_right_of‘
- Distance-based: ‘within_25m‘ (within 25 meters), ‘between_25m_and_40m‘ (between 25 and 40
meters), ‘between_40m_and_60m‘ (between 40 and 60 meters)

### Instructions:
1. Given the red bounding box, detect which OBJECT is in it.
2. Generate two RELATIONSHIPS between the detected OBJECT in the red bounding box and ego:
- One positional RELATIONSHIP (‘in_front_of‘, ‘to_left_of‘, ‘to_right_of‘)
- One distance-based RELATIONSHIP (‘within_25m‘, ‘between_25m_and_40m‘, ‘between_40m_and_60m‘)
3. Use the following structured format for your output.

### Output Template:
‘‘‘
[
(OBJECT, RELATIONSHIP, ego),
(OBJECT, RELATIONSHIP, ego),
]
‘‘‘

Using the above prompt format, please analyze the following image and generate the output
according to the template:
||*IMAGE*||
```

Figure 5.12: Query mode 3 base template.

from an object detection system, such as Detectron2 [195], Ground DINO [76], YOLO v11 [196], or YOLO World [197]. While this would make the pipeline's performance dependent on the detector's accuracy, existing object detection systems are becoming highly accurate [198].

Mode 4

This mode combines the specificity of yes/no questions with the targeted guidance of bounding boxes. In each query, we pair every bounding box and entity type with 6 yes/no questions regarding the entity's spatial relationships to the ego vehicle (one for each of the 6 relationships). Figure 5.13 shows the base prompt for this query mode, and Figure 5.14b shows an example where given a bounding box, we ask the VLM if the entity in the bounding box is in front of the ego vehicle. If the response is "Yes", the Response Aggregator adds the triplet to the final list of SG triplets. While query mode 4 is query-intensive, as it requires 6 separate queries for every detected entity

```
### Definitions:
1.  **ego**:  The vehicle holding the camera and capturing the scene (not visible in the
image).
2.  **OBJECT**:  Refers to one of the following:
- Person
- Bicycle
- Car
3.  **RELATIONSHIP**:  Refers to one of the following:
- Positional:  ‘in_front_of‘, ‘to_left_of‘, ‘to_right_of‘
- Distance-based:  ‘within_25m‘ (within 25 meters), ‘between_25m_and_40m‘ (between 25 and 40
meters), ‘between_40m_and_60m‘ (between 40 and 60 meters)

### Instructions:
Is the OBJECT in the red bounding box ||*QUESTION*||?
Answer yes/no.
||*IMAGE*||
```

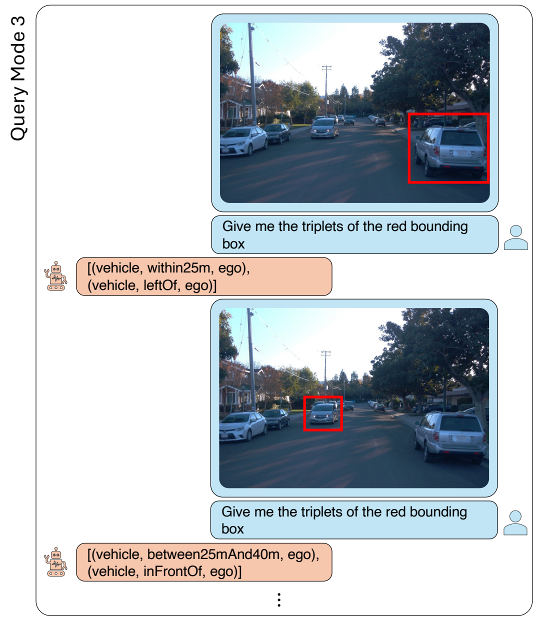
Figure 5.13: Query mode 4 base template.

in the scene, it is expected to reduce the likelihood of VLM errors and ensures that only relevant relationships are examined.

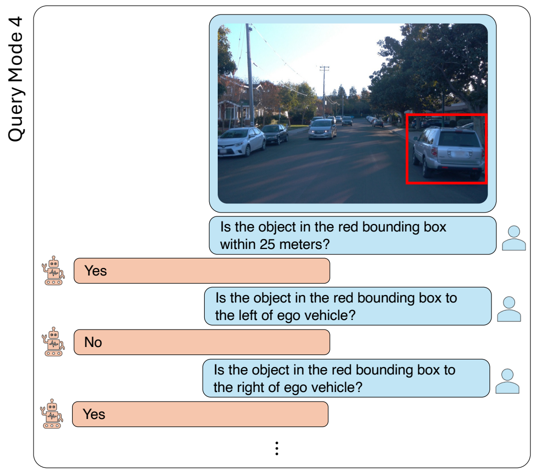
5.2.1.3 Limitations

The primary limitation of this approach lies in the granularity of the scene graphs. While the selected subjects (vehicles, persons, bicycles) and relations (relative position and distance) are sufficient for evaluating a set of safety-critical interactions, the graphs do not capture road topology, such as lane boundaries and intersections, the state of traffic lights, or traffic signs. This scope restricts the ability to define and monitor more complex safety properties that depend on these elements. Future work should therefore focus on extending the DriST dataset and vocabulary to incorporate additional contextual information, enabling the evaluation of a broader range of nuanced driving behaviors.

A second limitation concerns the computational efficiency of VLMs as SGGs. As we will show in Table 5.4, our fine-tuned LLaVA 1.5 models that achieved the best metrics, have an inference latency between 2.57 and 5.42 seconds per image depending on the query mode. These latencies, combined with the model size (7 billion parameters), present challenges for real-time deployment on resource-constrained AVs. Additionally, running inference on large VLMs incurs high computational


Query Mode 3
Give me the triplets of the red bounding box
[(vehicle, within25m, ego), (vehicle, leftOf, ego)]
Give me the triplets of the red bounding box
[(vehicle, between25mAnd40m, ego), (vehicle, inFrontOf, ego)]
⋮

(a) Query mode 3


Query Mode 4
Is the object in the red bounding box within 25 meters?
Yes
Is the object in the red bounding box to the left of ego vehicle?
No
Is the object in the red bounding box to the right of ego vehicle?
Yes
⋮

(b) Query mode 4

Figure 5.14: Examples of Query mode 3 and 4.

costs, both in terms of on-vehicle hardware requirements and, if deployed on cloud infrastructure, in terms of operational expenses. This computational overhead may limit the practical applicability of VLM-based SGGs for safety-critical real-time monitoring scenarios where sub-second latencies are required.

A third fundamental limitation arises from the system's restriction to single-frame, vision-only inputs. Because the model processes images independently, it lacks temporal consistency, which prevents the assignment of persistent identifiers to entities across frames or the derivation of dynamic states like velocity and acceleration. Furthermore, the exclusion of complementary sensor modalities, such as LiDAR or Radar, prevents the system from leveraging explicit depth data to ensure geometric consistency. Finally, the model's generalization is limited to the distribution of the training data, rendering it susceptible to out-of-distribution errors, where rare weather conditions or edge-case scenarios not represented in the training set may lead to incorrect predictions.

5.2.2 Evaluation

To assess the effectiveness of using VLMs as SGGs for autonomous driving, we set the following research question:

- RQ: How effective are VLMs and querying strategies at generating accurate SG triplets from driving scene images?

To answer it, we first detail the experimental setup, including the models and metrics used. We then present a quantitative analysis of VLM performance across the four query modes. Finally, we present a case study in Section 5.2.3 to demonstrate the practical utility of the best-performing model in a runtime monitoring application.

5.2.2.1 Setup

This section outlines the experimental setup for our evaluation. It begins by describing the diverse range of VLMs selected for comparison, which includes both general-purpose, state-of-the-art models and models specifically fine-tuned on the DriST dataset. It then formally defines the metrics—mean precision, mean recall, and mean F1 score—used to quantitatively measure the accuracy of the generated scene graphs against the ground truth.

Models

We selected a range of models that represent different configurations in both language modeling and vision encoding. A summary of the different VLMs, number of parameters, capabilities and citations is pressented in Table 5.3. We include GPT-4 Turbo (GPT-4-T) [199] as a reference for top-tier multimodal performance, setting a high baseline for model comparison, and RoadScene2Vec (RS2V) [200] as a traditional scene graph generator for the driving domain. As a foundational model, LLaVA 1.5 [80] combines the popular CLIP [75] image encoder with Llama 2 [81]. We also include LLaVA 1.6 Mistral [82] and LLaVA 1.6 Vicuna [82], both designed to handle higher image resolutions and fine-tuned on a more diverse dataset than LLaVA 1.5, while leveraging the latest advancements in their respective language models. Further diversity is introduced with PaliGemma [201], which leverages SigLIP [202] as an alternative vision encoder, potentially broadening the range of visual contexts the model can process.

To specifically evaluate spatial understanding, we also selected models like SpaceLLaVA [86], which was fine-tuned on spatial data. Similarly, the Cambrian Phi 3 [85] and Llama 3 [85] models incorporate the Spatial Vision Aggregator (SVA) to combine features from multiple vision encoders, thereby enabling more nuanced spatial reasoning through feature aggregation.

To enhance the spatial reasoning capabilities of VLMs specifically for autonomous driving scenarios, we experimented with using the DriST dataset to fine-tune and trained LoRAs [203] on LLaVA 1.5 (L1.5 Fine-tuned, L1.5 LoRA) in different query modes. We used 75% of the dataset for training, 5% for validation, and the remaining 20% for testing. While LLaVA 1.5 is somewhat dated, its repository benefits from extensive community support, and includes reliable scripts and documentation for efficient fine-tuning and LoRA training on GPUs. This model is also cost-effective; we fine-tuned our models using 8 A40 GPUs for up to 72 hours.

Model	**Params**	**Capabilities**	**Citations**
GPT-4 Turbo (GPT-4-T) [199]	175B	Top proprietary multimodal VLM	19,573
LLaVA 1.5 (L1.5) [80]	7B	CLIP [75] vision encoder with Llama 2 [81]	9,948
LLaVA 1.6 Mistral (L1.6 Mis) [82]	7B	Supports higher image resolution using grid splitting	3,720
LLaVA 1.6 Vicuna (L1.6 Vic) [82]	7B		
SpaceLLaVA [86]	7B	CLIP vision encoder and Llama 2 fine-tuned on spatial data	449
PaliGemma [201]	3B	SigLIP [202] vision encoder and Gemma 2B [204]	397
Cambrian Phi 3 (C-Phi3) [85]	3B	Multiple vision encoders with SVA and Phi 3 [205]	580
Cambrian Llama 3 (C-Llama3) [85]	8B	Multiple vision encoders with SVA and Llama 3 [88]	

Table 5.3: Summary of VLMs

Metrics

To quantify the performance of each VLM and query mode, we utilize mean precision@∞, mean recall@∞ [206], and mean F1@∞ as our primary metrics (henceforth, simply referred to as mean precision, mean recall, and mean F1). The @∞ notation indicates that the metrics are calculated over the entire set of predicted triplets, rather than being limited to a specific top-k value. We use

@∞ rather than a specific k-value because VLMs do not assign confidence scores to their predicted triplets, and our goal is to evaluate all triplets present in a scene even if they are repeated. Each model-query combination produces a list of predicted triplets, which we compare against a ground-truth list available in the DriST dataset. These lists are multisets to account for instances where the same triplet may appear multiple times within a scene, such as multiple pedestrians located to the right of the ego vehicle.

Let G_x denote the ground-truth multiset of triplets for a given scene x, and V_x denote the multiset of predicted triplets generated by the VLM V for the same scene x. We use $m_A(t)$ to refer to the count (or multiplicity) of triplet t in a multiset A. $m_A(t) = 0$ if t does not appear in multiset A.

Definition 5.2.1 (Intersection of multi-sets)**.** Let $\mathcal{T}$ be the set of all possible triplets. The intersection $G_x \cap V_x$ of multisets G_x and V_x is defined using the count $m(t)$ of each triplet t as,

$$\forall t \in \mathcal{T}.\ m_{G_x \cap V_x}(t) = \min(m_{G_x}(t), m_{V_x}(t))$$

Definition 5.2.2 (Mean precision and recall of VLM)**.** Let D be a set of images. If G_x and V_x are the ground-truth and predicted multiset of triplets for an image x and VLM V, the mean precision (P) and recall (R) of the VLM V with respect to D are defined as,

$$\mathrm{P} = \frac{1}{|D|} \sum_{x \in D} \frac{|G_x \cap V_x|}{|V_x|} \qquad \mathrm{R} = \frac{1}{|D|} \sum_{x \in D} \frac{|G_x \cap V_x|}{|G_x|}$$

For a scene x, precision measures the proportion of correct triplets among those predicted by the VLM V, while recall measures the proportion of ground-truth triplets correctly identified by the VLM V. The F1 score provides a harmonic mean of precision and recall. For each model, we compute the mean of these precision, recall, and F1 scores with respect to the test set.

5.2.2.2 RQ: Effectiveness of VLMs

To evaluate the models, we sampled 300 images at random from the DriST test split (see Table 5.2) for each of the three datasets. This process resulted in a total evaluation set of 900 images. Due

Model Name	**Query Mode**	**Time (s)**	**All Datasets**	**Kitti**	**Waymo**	**NuScenes**
Cambrian Llama 3	1	15.29	0.19	0.15	0.18	0.23
Cambrian Phi 3	1	9.85	0.26	0.31	0.26	0.21
GPT-4 Turbo	1	5.89	0.45	0.45	0.37	0.53
LLaVA 1.5	1	3.95	0.36	0.44	0.35	0.28
LLaVA 1.6 Mistral	1	3.15	0.36	0.35	0.38	0.35
LLaVA 1.6 Vicuna	1	3.65	0.25	0.26	0.27	0.23
PaliGemma	1	1.02	0.33	0.38	0.32	0.30
RoadScene2Vec	1	0.05	0.27	0.00	0.31	0.51
SpaceLLaVA	1	11.16	0.29	0.39	0.24	0.25
LLaVA 1.5 LoRA	1	2.58	0.65	**0.73**	0.66	0.55
LLaVA 1.5 Fine-tuned	1	2.57	**0.66**	0.72	**0.67**	**0.59**
Cambrian Llama 3	2	8.71	0.54	0.55	0.56	0.51
Cambrian Phi 3	2	8.20	0.52	0.51	0.56	0.49
GPT-4 Turbo	2	108.81	0.42	0.46	0.44	0.37
LLaVA 1.5	2	5.13	0.45	0.44	0.46	0.44
LLaVA 1.6 Mistral	2	8.93	0.50	0.47	0.52	0.49
LLaVA 1.6 Vicuna	2	8.25	0.45	0.35	0.48	0.50
PaliGemma	2	1.69	0.27	0.31	0.32	0.20
SpaceLLaVA	2	14.44	0.42	0.44	0.47	0.34
LLaVA 1.5 LoRA	2	4.84	0.67	0.69	0.69	0.61
LLaVA 1.5 Fine-tuned	2	4.81	**0.74**	**0.84**	**0.74**	**0.64**
Cambrian Llama 3	3	4.59	0.47	0.45	0.40	0.55
Cambrian Phi 3	3	4.01	0.45	0.42	0.37	0.56
GPT-4 Turbo	3	27.63	0.52	0.49	0.38	0.71
LLaVA 1.5	3	9.58	0.42	0.40	0.35	0.50
LLaVA 1.6 Mistral	3	5.31	0.45	0.44	0.39	0.53
LLaVA 1.6 Vicuna	3	5.60	0.38	0.29	0.34	0.50
PaliGemma	3	2.82	0.39	0.35	0.36	0.47
SpaceLLaVA	3	9.39	0.30	0.26	0.30	0.33
LLaVA 1.5 LoRA	3	4.35	**0.90**	**0.89**	0.90	**0.92**
LLaVA 1.5 Fine-tuned	3	4.30	0.89	0.87	**0.90**	0.88
Cambrian Llama 3	4	10.20	0.50	0.63	0.27	0.59
Cambrian Phi 3	4	9.19	0.25	0.17	0.19	0.37
GPT-4 Turbo	4	169.72	0.61	0.59	0.46	0.78
LLaVA 1.5	4	6.40	0.56	0.50	0.57	0.61
LLaVA 1.6 Mistral	4	26.91	0.52	0.45	0.51	0.61
LLaVA 1.6 Vicuna	4	9.72	0.55	0.50	0.54	0.63
PaliGemma	4	2.30	0.56	0.50	0.57	0.61
SpaceLLaVA	4	28.72	0.56	0.50	0.57	0.61
LLaVA 1.5 LoRA	4	5.44	0.81	0.78	0.84	0.79
LLaVA 1.5 Fine-tuned	4	5.42	**0.93**	**0.93**	**0.93**	**0.93**

Table 5.4: F1 scores and times using 4 query modes for Kitti, Waymo, NuScenes.

to the operational cost associated with querying GPT-4 Turbo, we utilized a smaller subset for this specific model. This subset was created by further sampling 30 images from each 300 image set, leading to a total of 90 images for the GPT-4 Turbo evaluation.

The results from our evaluation (Table 5.4) highlight several key insights into the performance of VLMs when using various querying approaches to capture spatial relationships triplets in autonomous driving scenes. An extended table with additional information is provided in the appendix. Notably, the family of fine-tuned LLaVA 1.5 models achieved the highest performance across query modes, outperforming all other models, including GPT-4 Turbo. This outcome underscores the effectiveness of DriST in enhancing VLMs capabilities to capture spatial relationships for driving scenarios. Furthermore, the LoRA-trained models—optimized for each query mode—also performed competitively, especially in Mode 3, where LoRA-trained LLaVA 1.5 marginally outperformed the fully fine-tuned variant.

The choice of query mode significantly impacted model performance, as observed in the total column of the results table. The incremental improvement from Mode 1 to Mode 2, and similarly from Mode 3 to Mode 4, suggests that structuring the prompts differently can help the model better capture the desired spatial triplets. Specifically, the more targeted querying approaches used in Modes 2 and 4 lead to better identification of spatial relationships, though they come at the expense of longer processing times due to the increased number of queries. The role of query mode in enhancing VLM performance points to the potential of exploring additional querying strategies to further optimize spatial relationship extraction.

A closer examination of the results across datasets reveals some variability, particularly in Modes 1 and 2. The best models generally performed worse on nuScenes, with lower F1 scores compared to KITTI and Waymo. This drop in performance likely arises from the poorer quality of some nuScenes images, which can be blurry or low-light, making it challenging for VLMs to accurately identify entities and their spatial relationships. Interestingly, in Modes 3 and 4, where bounding boxes were provided, this variability across datasets largely disappeared, with the best-performing models achieving consistent F1 scores across all three datasets. This suggests that in challenging visual conditions, the presence of bounding boxes assists the VLMs by focusing their attention on specific entities, thereby mitigating the impact of image quality.

Additionally, the poor performance of RoadScene2Vec, particularly on KITTI where it scored an F1 of 0, further highlights the sensitivity of some SGGs to dataset-specific camera configurations. RoadScene2Vec 's reliance on camera information in its configuration file means that its performance is notably affected by differences in field of view, as KITTI images are wider than those in nuScenes and Waymo. Consequently, RoadScene2Vec performed relatively better on the other two datasets.

In terms of computational efficiency, the times recorded for each model and query mode reveal important trends. For Mode 1, RoadScene2Vec achieved the fastest times given its architecture, but its F1 score was less than a third of the best-performing model, suggesting a trade-off between speed and accuracy.

The fine-tuned VLM and LoRA models struck a balance between speed and F1 score, achieving both the best times and highest accuracy in this mode. This efficiency is likely due to the VLMs generating concise text with minimal repetition or unnecessary explanations, which kept generation time low. In Mode 2, despite the 18 individual questions asked, the fine-tuned VLM and LoRA models once again proved the fastest, as the responses only required simple yes/no answers. In contrast, other models were more verbose in their responses, which increased generation times. In Mode 3, Cambrian Phi3 demonstrated the fastest processing times among all models, though its F1 score was only half that of the top-performing fine-tuned VLM. Following Cambrian Phi3, the fine-tuned VLM and LoRA models were also relatively fast in this mode. Cambrian Phi3's shorter inference times likely stem from its smaller parameter size, which, combined with concise descriptions of the triplets, resulted in efficient processing. Finally, in Mode 4, the fine-tuned VLM and LoRA models once again emerged as the fastest, providing similar advantages in processing as in Mode 2, where concise yes/no responses minimized generation time.

5.2.2.3 Threats to validity

Our evaluation is subject to two main threats to validity. The primary threat is to external validity, concerning the generalizability of our findings. The evaluation relies on the DriST dataset, which, while composed of diverse scenes from KITTI, Waymo, and nuScenes, may not capture the full spectrum of driving conditions, such as extreme weather (e.g., heavy snow, dense fog) or highly unusual road events. Consequently, the high performance of our fine-tuned models may not directly translate to these out-of-distribution scenarios. Furthermore, our analysis is constrained to the vocabulary of objects and spatial predicates defined within the DriST dataset. The models' ability to recognize a broader or more nuanced set of relationships remains untested.

Another threat relates to construct validity and the idealizations in our experimental setup. Our methodology assumes the ground-truth annotations in the DriST dataset are entirely accurate and comprehensive. However, manual annotation is susceptible to human error or subjective interpretation, which could introduce noise into our evaluation metrics. Furthermore, for Modes 3 and 4, our use of ground-truth bounding boxes represents a significant simplification. In a real-world deployment, these bounding boxes would be generated by an upstream 2D object detector, which is subject to its own errors (e.g., missed detections, inaccurate localizations). The performance of our models in these modes should therefore be considered an upper bound, as errors from a perception module would inevitably propagate and degrade the VLM's accuracy. Additionally, the process of converting the VLM's natural language output into structured triplets relies on parsing. While effective for quantitative analysis, this step could misinterpret syntactically correct but stylistically unanticipated responses, potentially penalizing a model for its phrasing rather than its underlying spatial understanding. Finally, the rapid evolution of VLM architectures means that our conclusions are inherently tied to the specific models evaluated; future models with different architectures may exhibit different performance characteristics and challenges.

5.2.3 Case study

This case study demonstrates the practical utility of VLMs as SGGs in real-world runtime monitoring by assessing how effectively a sequence of SGs, generated by the VLM from individual images, can be used to monitor temporal safe driving properties. To perform this assessment, we compare the property violations identified by the monitor when using VLM-predicted SGs against the violations identified when using the ground-truth SGs. This comparison allows us to quantify the VLM's real-world applicability using the metrics defined in Section 5.2.3.1. For this analysis, we employ the best-performing VLM from Section 5.2.2.2 (LLaVA 1.5 Fine-tuned with query mode 4) to check the three properties illustrated in Table 5.5.

5.2.3.1 Setup

Driving Sequences

This case study analyzes the complete Waymo validation (5%) and test (20%) splits, as detailed in Table 5.2. This combined dataset consists of 199 sequences containing over 39k images (frames) in total. Although the validation split was used after each training epoch to report metrics, it was not used for gradient updates and did not alter the model's loss. Therefore, both splits represent data on which the model's parameters were not trained, making them appropriate for this case study evaluation. These sequences were collected in 3 cities: San Francisco, Mountain View, and Phoenix, at different times of the day, containing 196 frames on average. These frames contain 67,566 vehicles, 1,698 bicycles, and 22,027 persons. We leverage the ground truth triplets from these 199 sequences to check the properties and compare the violations detected in the ground truth to those identified using the VLM's predictions.

Properties

The SGs generated by VLMs capture key spatial relationships but lack the granularity to describe complex road layouts or traffic signal states. Consequently, the properties selected for this case study, presented in Table 5.5, are a simplified subset of the more comprehensive specifications found in Table 3.4.

Property	English Description	Definition
ϕ_1	Brake if an object is within 25m and speed is $\geq$ 25mph.	$G(anythingWithin25m \wedge egoSpeed \geq 25mph \wedge X(anythingWithin25m \wedge egoSpeed \geq 25mph) \rightarrow X\ isBraking)$
ϕ_2	Brake if an object is between 25-40m and speed is $\geq$ 35mph.	$G(anythingBetween25\_40m \wedge egoSpeed \geq 35mph \wedge X(anythingBetween25\_40m \wedge egoSpeed \geq 35mph) \rightarrow X\ isBraking)$
ϕ_3	Brake if an object is between 40-60m and speed is $\geq$ 45mph.	$G(anythingBetween40\_60m \wedge egoSpeed \geq 45mph \wedge X(anythingBetween40\_60m \wedge egoSpeed \geq 45mph) \rightarrow X\ isBraking)$

Table 5.5: Safe driving properties analyzed in the case study.

To evaluate these properties using the VLM's output, we define the propositions as a combination of the specific scene graph triplets. In Table 5.5, the terms *anythingWithin25m*, *anythingBetween25_40m*, and *anythingBetween40_60m* for ϕ_1, ϕ_2, and ϕ_3, respectively serve as shortnames for the disjunction of all relevant entities at those specific ranges. For example: *anythingWithin25m* $= <$ *vehicle, within25m, ego* $> \vee <$ *bicycle, within25m, ego* $> \vee <$ *person, within25m, ego* $>$. This disjunction means that the proposition *anythingWithin25m* is true if any of the three entity types (vehicle, bicycle, or person) is detected within 25 meters of the ego vehicle.

Metrics

We present the results in terms of the following metrics, which compare the monitor's findings using VLM-predicted SGs against the ground truth: *True Violations (True Positives)*, cases where a property violation occurred in the ground truth and was also identified by the monitor using the VLM-predicted SGs; *False Violations (False Positives)*, cases where the monitor, using VLM-predicted SGs, identified a property violation that did not occur in the ground truth; *Missed Violations (False Negatives)*, cases where a property violation occurred in the ground truth but was not identified by the monitor using the VLM-predicted SGs; finally, *True Non-Violations (True Negatives)*, cases where no violation occurred in the ground truth and none was identified by the monitor using the VLM-predicted SGs.

5.2.3.2 Results

Table 5.6 summarizes the results. The monitor successfully identified 74% true violations (TP) for ϕ_1, 78% for ϕ_2, and 100% for ϕ_3; while reporting 1% of false violations (FP) for ϕ_1, 1% for ϕ_2; and none for ϕ_3. The low number of false violations (FP) suggests that the monitors are precise in their predictions, and the low number of missed violations (FN) indicates that the monitors do not miss many entities relevant to the properties.

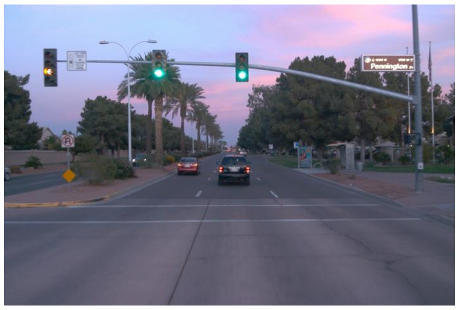
Furthermore, the high count of true non violations (TN) suggests that the monitor correctly identified sequences without violations. We note that 15/179 (8%) in ϕ_1, 4/188 (2%) in ϕ_2, and 4/193 (2%) in ϕ_3 are the cases where the speed of the ego vehicle is higher than the threshold. For those cases, it means that the VLM accurately predicted the relevant triplets, without leading to false violations. The remaining true non-violation cases (e.g., 164 for ϕ_1, 184 for ϕ_2, and 189 for ϕ_3)

		Ground Truth	
		Violation	Non-Violation
VLM	Violation	True Positives (TP) $\phi_1 = 14$ $\phi_2 = 7$ $\phi_3 = 6$	False Positives (FP) $\phi_1 = 2$ $\phi_2 = 2$ $\phi_3 = 0$
	Not Violation	False Negatives (FN) $\phi_1 = 5$ $\phi_2 = 2$ $\phi_3 = 0$	True Negatives (TN) $\phi_1 = 179$ $\phi_2 = 188$ $\phi_3 = 193$

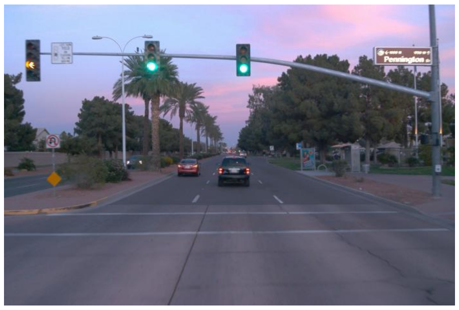
Table 5.6: Case study results.

are sequences where the ego vehicle's speed was below the property's threshold. In these instances, the property is trivially satisfied as the speed condition is not met, and no violation is reported, regardless of the VLM's object predictions.

Figure 5.15 shows an example of a true violation (TP) of ϕ_2, where there is a vehicle between 25m and 40m (black SUV), the ego speed is 44 mph and the ego vehicle has a positive acceleration. In contrast, Figure 5.16 shows an example of a missed violation (FN), where there is a vehicle within 25m and 40m, yet the VLM did not detect that there is one. In this case, the ground truth indicates that the car is between 25m and 40m, while the VLM predicts that the car is between 40m and 60m, indicating that the it is not between 25m and 40m. This is a subtle miss-classification from the VLM given that the car is close to the range limit, around 40 meters.

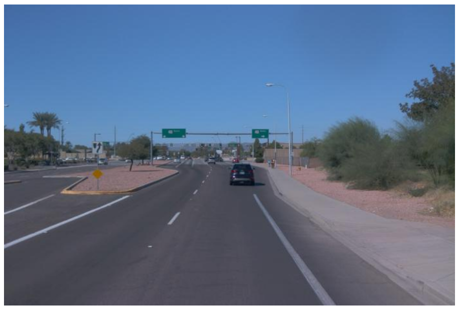


Time Step 1
Speed = 44 mph (egoSpeed >= 35 mph ≡ True)
Acceleration = 0.54 m/s^2 (isBraking ≡ False)
Ground Truth
vehicle_between_25_40m ≡ True
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingWithin25m ≡ True)
VLM Prediction
vehicle_between_25_40m ≡ True *
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingWithin25m ≡ True)

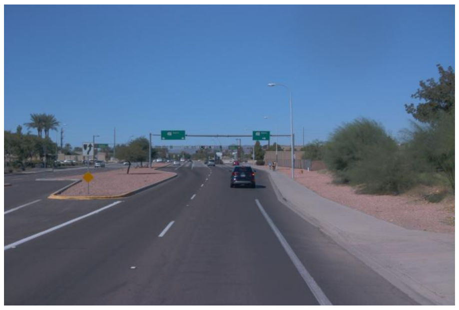


Time Step 2
Speed = 44 mph (egoSpeed >= 25 mph ≡ True)
Acceleration = 1.04 m/s^2 (isBraking ≡ False)
Ground Truth
vehicle_between_25_40m ≡ True
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingWithin25m ≡ True)
VLM Prediction
vehicle_between_25_40m ≡ True *
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingWithin25m ≡ True)

Figure 5.15: ϕ_2 True Violation (TP).



Time Step 1
Speed = 43 mph (egoSpeed >= 35 mph ≡ True)
Acceleration = -0.64 m/s^2 (isBraking ≡ True)
Ground Truth
vehicle_between_25_40m ≡ True
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingBetween25_40m ≡ True)
VLM Prediction
vehicle_between_25_40m ≡ False *
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingBetween25_40m ≡ False)



Time Step 2
Speed = 43 mph (egoSpeed >= 35 mph ≡ True)
Acceleration = 0.05 m/s^2 (isBraking ≡ False)
Ground Truth
vehicle_between_25_40m ≡ True
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingBetween25_40m ≡ True)
VLM Prediction
vehicle_between_25_40m ≡ False *
bicycle_between_25_40m ≡ False
person_between_25_40m ≡ False
(anythingBetween25_40m ≡ False)

Figure 5.16: ϕ_2 Missed Violation (FN).

5.2.4 Summary

This section explored the potential of VLMs to address the limitation of the monitoring approach from Section 5.1, which relied on privileged simulator data. To move towards real-world application, this work investigated using VLMs as SGGs, which included developing the DriST dataset and evaluating query strategies. The evaluation showed that VLMs fine-tuned on DriST, specifically the LLaVA 1.5 Fine-tuned model using Mode 4, can generate SGs with high F1 score. The case study supported this, showing that a monitor using VLM's generated SGs was able to identify 74% to 100% of true safety violations on Waymo sequences. This work thus establishes the feasibility of real-world monitoring using SGs from camera images. However, the VLM's inference latency and the limited information encoded in the SGs (e.g., lacking road topology) present a limitation for using this approach in proactive correction. The next section addresses this by introducing a faster SGG that produces richer SGs and a corrector mechanism.

5.3 Correcting AV behavior to Ensure Safe Driving Property Conformance

The previous sections established the feasibility of runtime monitoring, first in simulation and subsequently in real-world settings by leveraging VLMs to interpret camera images. However, the VLM-based approach, while a crucial step towards deployment, is hampered by significant inference latency and the limited semantic information contained in the generated SGs, e.g., not including road lanes, traffic lights or signs. These shortcomings render it insufficient for the ultimate goal of proactive correction, where the system must not only detect but also prevent impending safety violations in real-time. Transitioning from passive monitoring to active correction therefore demands a shift towards a more performant and information-rich SGs.

This section introduces Monitoring for Property Conformance (M4PC), an approach that transitions from passive monitoring to active correction. The approach is designed to enforce safe driving property conformance through minimal, real-time adjustments to the ADS's control outputs. This capability is distinct from training approaches, like T4PC described in Section 4.2, which attempt to teach the model to adhere to rules before deployment. While effective at improving conformance with safe driving properties, its applicability is limited to end-to-end differentiable ADS, and it requires access to internal DNN weights, making them unsuitable for modular or closed-source ADS. Furthermore, the improvements gained during training are bounded by the training distribution and may not prevent failures arising from unseen scenarios encountered in the real world. Therefore, a runtime correction mechanism is needed to actively mitigate potential failures during operation. We explicitly design this approach to be architecture-agnostic, treating the underlying system as a black box. This flexibility enables M4PC to improve safety on any autonomous driving architecture, ranging from end-to-end ADS to traditional modular pipelines that rely on a combination of programmed and learned components. To that end, this section first details the M4PC approach and subsequently presents its evaluation, demonstrating its effectiveness in proactively preventing property violations.

5.3.1 Approach

We propose M4PC, an approach to monitor and correct AVs. This approach ensures continued operation by specifying safety properties over SGs. To achieve this, M4PC leverages a novel SGG to evaluate property preconditions at runtime. Based on this evaluation, the system efficiently performs in situ corrections, ensuring that the control outputs produced by the system abide by the required postcondition. The pipeline, shown in Figure 5.17, consists of two major components: (i) SGG from sensor inputs, and (ii) Correction Mechanism for runtime correction. Together, these components enable the system to detect and mitigate potential failures before violating a driving rule.

As depicted in Figure 5.17, the M4PC approach begins with the SGG processing raw sensor inputs from the world. This generator utilizes UniAD, a DNN, to obtain lane segmentation and 3D bounding boxes which are then passed to RelExtract to construct an SG. This SG is fed into the Correction Mechanism. Within this, the Monitor evaluates the SG against the preconditions ϕ_x derived from the specified properties Φ to identify the active properties, i.e., those whose preconditions are satisfied. The Constraints Intersection module then computes the output constraints required to satisfy the

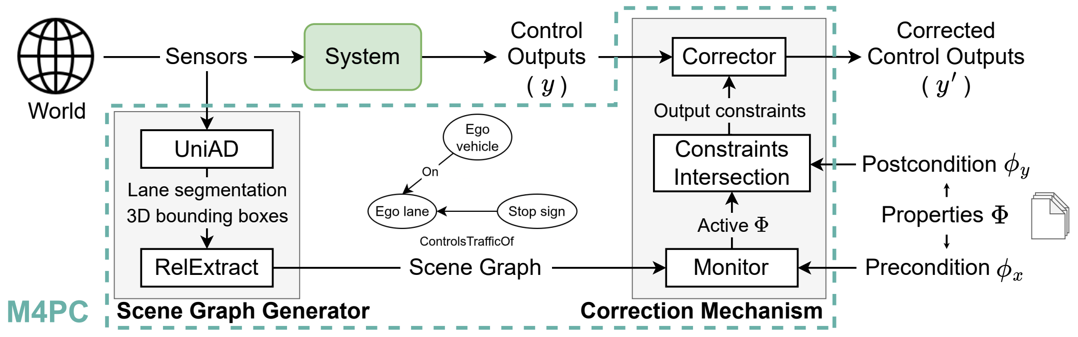


Figure 5.17: M4PC Approach Diagram.

postconditions ϕ_y of all these active properties. Concurrently, the autonomous System processes sensor data to produce its own control outputs (y). Finally, the Corrector compares the system's outputs with the computed constraints, issuing corrected control outputs (y') that adhere to all active postconditions.

5.3.1.1 Properties

M4PC support properties in RFOL expressed as precondition and postcondition pairs, $\phi_{\mathcal{X}} \implies \phi_{\mathcal{Y}}$, as introduced in Section 3.2, consistent with the format used in Section 4.2. We assume this set of properties is internally consistent, utilizing the definitions and automated checks established in Section 4.2.1.2. While the monitoring approach in Section 5.1 supported temporal specifications using SGL and LTL$_f$, M4PC focuses on properties defined over a single time step. This is because the postcondition of an RFOL property explicitly defines the allowable output region for the current input, enabling immediate correction. In contrast, correcting a temporal property violation is non-trivial as different actions may need to be taken depending on the monitor's current state, which varies based on the history of events.

5.3.1.2 Scene Graph Generator (SGG)

M4PC translates raw sensor observations into a structured semantic representation through a two-stage pipeline combining perception and relational reasoning.

The first subcomponent is meant to identify objects and lane boundaries in images. In this work we use **UniAD** [207], a foundational model for autonomous driving that performs lane segmentation and 3D object detection. This specialized model is employed to overcome the limitations of using VLMs as SGGs, which include significant inference latency and an inability to capture necessary road topology and signange elements like traffic lights or signs, as discussed in Section 5.2.1.3. UniAD processes multi-camera images through a unified transformer-based architecture to generate 3D bounding boxes for objects (e.g., vehicles, pedestrians) and segmentation maps for the drivable area. Unlike modular pipelines that process tasks in isolation, this architecture shares feature representations across tasks, ensuring that detected objects and lane boundaries are spatially and temporally aligned. This alignment is crucial for reasoning as it allows the system to accurately associate scene elements, such as determining which lane an object occupies or which traffic light controls it. Furthermore, the temporal consistency ensures that these associations remain stable over time, preventing unintended changes in the perceived environment due to sensor noise. As newer and more accurate perception models emerge, the M4PC architecture enables model updates as long as the entities required by the target properties are properly captured.

The second subcomponent, **RelExtract**, systematically transforms the previous perception outputs into a structured scene graph representation. Our subcomponent first converts the 3D bounding boxes and lane information into a unified Bird's Eye View (BEV) representation, providing a consistent spatial reference frame for relationship extraction. It then constructs the SG by instantiating each detected object as a node and establishing semantic edges that encode driving-relevant relationships. Traffic control elements, such as stop signs and traffic lights, are linked to the lanes they regulate, while mobile entities, including vehicles and pedestrians, are associated with their respective lanes when applicable. Spatial relationships (e.g., inFrontOf, toRightOf, toLeftOf) and distance-based relationships (e.g., within7m, between7_10m) are computed relative to the ego vehicle's position. This structured approach ensures that the scene graph captures all relationships necessary for downstream reasoning, providing a compact, interpretable, and semantically rich structure that enables the evaluation of driving properties and supports the correction mechanism.

5.3.1.3 Correction Mechanism

The **Monitor** module takes as input the current scene graph and the preconditions from the set of target properties, where each property encodes a driving rule. The Monitor evaluates each of the preconditions over the scene graph to identify the set of active properties, Φ_{active}. To do so, the Monitor leverages the mechanisms from prior work described in Section 5.1 to determine whether the precondition holds over the graph. Given the SG, sg, $\Phi_{active} = \{\phi \mid \phi \in \Phi, \phi_{\mathcal{X}}(sg)\}$.

Once the set of active properties has been identified, the **Constraints Intersection** module takes the set of active properties and identifies the intersection of their postconditions to determine the required output constraints, $C = \bigwedge_{\phi \in \Phi_{active}} \phi_{\mathcal{Y}}$. This process is illustrated in Figure 5.18. From Section 5.3.1.1, C must be non-empty by construction of Φ. This defines the region of the output space which satisfies all active postconditions; M4PC will then ensure that the system's output falls within this region.

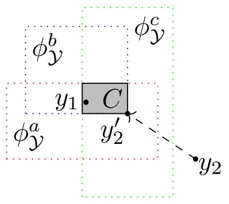


Figure 5.18: Constraint Intersection, C, for $\phi_{\mathcal{Y}}^{a}$, $\phi_{\mathcal{Y}}^{b}$, $\phi_{\mathcal{Y}}^{c}$ shaded in gray and the correction applied to outputs y_1 and y_2.

Independent of M4PC, the system under monitoring (SuM) uses its internal mechanism to compute its intended control output, y. The **Corrector** module compares the output to the constraint intersection, C, and performs a minimal correction to produce a new control output, y'. If the output already lies within this region, then the postcondition is satisfied and no correction is needed. This is illustrated with y_1 in Figure 5.18; there is no separate y_1' since $y_1 \in C$. However, if the output does not lie within C, then the Corrector identifies the nearest point within C and corrects the output to that point. This ensures that the new output satisfies the postcondition and that the correction applied is minimal, preserving the SuM's intended behavior as much as possible. This is illustrated with y_2 in Figure 5.18; the SuM's intended control output lies outside of the region,

$y_2 \notin C$, and must be corrected to $y'_2 \in C$. This process is inspired by the training-time correction mechanism utilized by T4PC, adapted for usage in our runtime correction framework. We remark that computing the constraint intersection is efficient; however, for large numbers of properties, the intersections of all possible combinations of properties could be precomputed and cached for additional runtime performance. This new output is then used to control the actions of the SuM. The corrected output is guaranteed to meet all active postconditions.

5.3.1.4 Limitations

Although we constructed M4PC to address lack of generalization of other approaches, it has some other limitations.

First, the quality of the constructed scene graphs depends on the precision and recall of the underlying perception model, UniAD. Missed detections or false positives in entities or lane segmentation can lead to incomplete or incorrect scene graphs, which in turn may prevent the monitor from detecting violations or trigger unnecessary corrections. Additionally, UniAD is a deep neural network that is not optimized for real-time inference limiting the system's ability to operate at high rates required for real-world deployment. Future work will explore more efficient models to enable faster, scalable runtime monitoring and correction.

Second, the approach shares limitations with T4PC, as described in Section 4.2.1.5. Specifically, the approach supports only Hyper-Rectangular (HR) constraints and enforces properties defined over a single time step. This restriction prevents M4PC from correcting violations that rely on a sequence of events, such as right-of-way negotiation. Correcting such temporal violations is non-trivial because, unlike the properties supported by M4PC where the postcondition explicitly defines a valid output region, the necessary corrective action varies depending on the monitor's internal state and the history of the execution. Consequently, future work should investigate approaches to map these monitor states to specific output constraints to enable the enforcement of complex temporal driving behaviors.

5.3.2 Evaluation

This section presents a comprehensive evaluation of M4PC designed to assess its effectiveness and efficiency as a runtime correction mechanism for autonomous driving systems. The evaluation is structured around two primary research questions:

1. *How effective is M4PC at reducing property violations and improving driving performance across different autonomous system architectures?*

2. *What is the trade-off between M4PC's runtime performance and its effectiveness at ensuring property conformance?*

To answer these questions, we first detail the experimental setup, including the systems, properties, and metrics used. We then present a detailed analysis corresponding to each research question.

5.3.2.1 Setup

This section outlines the setup for our evaluation. We first introduce the three distinct autonomous driving systems chosen for evaluation, which represent both end-to-end and modular architectural paradigms. This selection is intended to demonstrate that M4PC operates as a black-box approach, applicable to any system regardless of its internal architecture. We then briefly describe the formal properties used to assess conformance, which are the same as to those detailed in Table 4.6, the simulation environment, and the metrics for quantifying performance.

Systems

We evaluate three autonomous driving systems. The first two, TCP [181] and Interfuser [182], have been widely adopted in the literature as strong baselines [158] and were also used as the units of analysis in Section 4.2. We note that unlike T4PC, which required access to model weights and the approximation of non-DNN components (e.g., Interfuser's coded controller) with neural networks to enable property loss finetuning, M4PC is entirely system-independent. It directly accommodates the system as built, sparing developers from the non-trivial effort of approximating components and finetuning models. The third system is Pylot [208], a popular autonomous vehicle platform with over 500 GitHub stars, designed for developing and evaluating autonomous driving components—such as perception, traffic lights and signs prediction, and planning—using the CARLA simulator. We selected Pylot because it is representative of many autonomous systems that employ a modular architecture, with distinct perception, planning, and control layers, making it a suitable smaller-scale alternative to comprehensive platforms like Autoware [209] and Apollo [210]. Importantly, Pylot's modular design provides a crucial contrast to TCP and Interfuser, as it does not rely on end-to-end deep neural networks, thus allowing us to evaluate our black-box approach across different architectural paradigms in autonomous driving systems.

Techniques

For two of the three systems under test, TCP and Interfuser, we consider the finetuned baselines created in Section 4.2 (denoted as *TCP Base* and *Interfuser Base*) and the corresponding T4PC models trained with property loss (denoted as *TCP T4PC* and *Interfuser T4PC*). For Pylot, we evaluate only the original system (referred to as *Pylot Base*). We then apply our new approach, M4PC, to all three baseline systems. While T4PC is the state-of-the-art for property-driven conformance, it requires costly retraining of machine-learned components and cannot be applied to modular, heterogeneous systems like Pylot. In contrast, M4PC operates at runtime without retraining or model access, treating the SuM as a black-box, making it broadly applicable across both end-to-end and modular architectures. We evaluate two instances of M4PC: one using ground-truth SGs extracted directly from the simulator using a CARLA plugin described in Section 3.1.3 (denoted *M4PC (GT)*), and another using M4PC's novel scene graph generator based on UniAD (denoted *M4PC (SGG)*). Additionally, we evaluate these same two M4PC instances applied to the T4PC-trained models for TCP and Interfuser (denoted as *T4PC + M4PC (GT)* and *T4PC + M4PC (SGG)*), which allows for an assessment of combining the two techniques. Regarding model selection, while T4PC trained five models per configuration to capture training variance, we select one model at random for each case (one finetuned baseline and one T4PC model) to serve as the baselines. By selecting one model instance, we treat the ADS as a specific deployment candidate, and rely on multiple simulation runs to control for environmental variance consistent with T4PC. As a result, our reported numbers do not exactly match those in the T4PC paper, but the comparison remains valid since all techniques are evaluated under the same conditions.

Properties

To evaluate the effectiveness of M4PC, we employ a set of six driving properties ($\phi_1 - \phi_4$ and $\phi_7 - \phi_8$) formalized in RFOL and summarized in Table 4.6. These properties capture common safety and traffic rule constraints, such as obeying traffic lights, stopping at stop signs, and avoiding collisions with leading vehicles. Following the setup in Section 4.2.3, we apply ϕ_1–ϕ_4 to Interfuser, since these rules cover interactions with traffic lights and surrounding vehicles, which are the main failure modes reported for this system. For TCP, we adopt ϕ_7–ϕ_8, as these rules were already identified in T4PC to effectively characterize the stop-sign related behaviors where TCP exhibits violations. Finally, for Pylot [208], we use the combination of ϕ_1, ϕ_2, ϕ_4 and $\phi_7 - \phi_8$, because the original system is prone to colliding with other vehicles, running red lights, and missing stop signs.

Simulation and Metrics

We evaluate all systems in the CARLA simulator using ten routes from Town 05, a map excluded from the training and validation of the tested models, with all features presented in Section 5.1.2.1 - Common Execution Platform. To account for stochastic variation in simulation, each system executes all ten routes five times. Performance is measured using the official CARLA leaderboard metrics, which combine safety and task completion aspects of driving. In particular, we report the average *driving score*, a composite measure that accounts for route completion and penalizes infractions, and the average *infraction score* for each relevant infraction type, which quantifies the severity and frequency of rule violations.

5.3.2.2 RQ#1: Effectiveness of M4PC

To answer this question, we run all instances of M4PC, evaluating its performance across different configurations and systems. We note that for both TCP and Interfuser, Base and T4PC models, we show the results computed in the study in Section 4.2.3. Table 5.7 reports the results for TCP, Interfuser, and Pylot on all techniques.

TCP

M4PC (GT) achieves an average driving score of 87.16, outperforms both the TCP Base (76.86)—

Treatment	Driving Score ↑	Collision Pedestrians ↓	Collision Vehicles ↓	Red Light Infraction ↓	Route Timeout ↓	Stop Sign Infraction ↓	Vehicle Blocked ↓
TCP							
Base	76.86±23.43	0.00±0.00	0.10±0.36	0.06±0.24	0.00±0.00	1.10±1.23	**0.00±0.00**
T4PC	78.23±21.76	0.00±0.00	0.12±0.33	**0.00±0.00**	0.00±0.00	0.94±1.13	0.02±0.14
M4PC (GT)	87.16±24.76	0.00±0.00	**0.04±0.20**	0.06±0.24	0.00±0.00	0.12±0.33	0.08±0.27
M4PC (SGG)	79.99±27.97	0.00±0.00	0.22±0.55	0.06±0.31	0.00±0.00	0.20±0.40	0.08±0.27
T4PC + M4PC (GT)	**95.31±15.38**	0.00±0.00	0.08±0.34	**0.00±0.00**	0.00±0.00	**0.02±0.14**	0.02±0.14
T4PC + M4PC (SGG)	80.73±24.53	0.00±0.00	0.22±0.58	0.06±0.24	0.00±0.00	0.50±0.81	0.04±0.20
Interfuser							
Base	59.41±32.28	0.26±0.49	0.64±0.88	0.42±0.67	0.06±0.24	0.00±0.00	**0.00±0.00**
T4PC	64.05±35.47	0.18±0.39	0.72±1.07	0.28±0.50	0.18±0.39	0.00±0.00	**0.00±0.00**
M4PC (GT)	**69.74±32.42**	0.10±0.36	**0.34±0.59**	**0.04±0.20**	0.02±0.14	0.00±0.00	0.26±0.44
M4PC (SGG)	67.84±30.98	**0.08±0.27**	0.40±0.73	0.38±0.73	**0.00±0.00**	0.00±0.00	0.14±0.35
T4PC + M4PC (GT)	63.24±36.80	0.12±0.39	0.44±0.67	0.14±0.35	0.10±0.30	0.00±0.00	0.24±0.43
T4PC + M4PC (SGG)	66.50±31.24	0.12±0.33	0.56±0.86	0.32±0.51	0.02±0.14	0.00±0.00	0.06±0.24
Pylot							
Base	68.96±26.70	0.00±0.00	0.32±0.65	0.54±0.86	**0.00±0.00**	0.44±0.54	0.02±0.14
M4PC (GT)	**80.53±24.03**	0.00±0.00	**0.20±0.45**	**0.38±0.64**	0.12±0.33	**0.00±0.00**	**0.00±0.00**
M4PC (SGG)	75.38±27.05	0.00±0.00	0.24±0.48	0.52±0.99	0.04±0.20	0.24±0.43	**0.00±0.00**

Table 5.7: TCP, Interfuser, and Pylot results for RQ#1. Values in bold are the best for each system.

deemed significant by an ANOVA with $p = 0.0504$ with Bonferroni correction—and TCP T4PC (78.23). This improvement stems primarily from a reduction in stop sign infractions, which decrease to 0.12 on average under M4PC (GT) from 1.10 in the TCP Base (89% decrease) and 0.94 in TCP T4PC (87% decrease). The SGG-based variant of M4PC has an average driving score of 79.99 and an average stop sign infractions of 0.2, a decrease of 82% and 79% compared to TCP Base and TCP T4PC. While M4PC (SGG) effectively enforces stop sign conformance through properties ϕ_7 and ϕ_8, it introduces more vehicle collisions than other methods. This trade-off reflects our deliberate design choice to focus on specific traffic rule violations rather than a limitation of M4PC itself. The collision increase could be mitigated by incorporating additional collision-prevention properties, which we omitted here to ensure fair comparison with T4PC's property set.

When combining T4PC with M4PC (GT), the driving score reaches 95.31, with stop sign infractions reduced to 0.02, demonstrating the effect of combining training-time and runtime correction. On the other hand, the SGG variant (T4PC + M4PC (SGG)) shows 1% driving score improvement over M4PC (SGG); but is still able to reduce the stop sign infractions from 0.94 under TCP T4PC to 0.5 (47% decrease). However, it has a higher stop sign infractions than M4PC (SGG) alone, reflecting a trade-off in combining the two approaches. The marginal improvement is driven by the difference in vehicle blocked infractions; whereas M4PC (SGG) incurred this infraction 5 times with an average route completion of 18%, T4PC + M4PC reduced this count to two with a higher completion rate of 39%.

Interfuser

Compared to the improvements observed with M4PC on TCP, M4PC on Interfuser showed smaller gains. M4PC (GT) achieves the highest driving score (69.74), an improvement over both the Base (59.41) and T4PC (64.05). This score is supported by reductions in several infractions. Compared to the Interfuser Base, M4PC (GT) reduced pedestrian collisions by 61% (to 0.10), vehicle collisions by 47% (to 0.34), and red light infractions by 90% (to 0.04). The reductions compared to Interfuser T4PC were 44%, 52%, and 86%, respectively. The SGG variant, M4PC (SGG), achieves a score of 67.84. It reduced pedestrian collisions to 0.08 (a 69% decrease from Base, 55% from T4PC) and vehicle collisions to 0.40 (a 37% decrease from Base, 44% from T4PC). However, its red light infractions (0.38) represented only a 9% decrease from Base and were a 35% increase compared to Interfuser T4PC. These results demonstrate that M4PC improves the overall driving score and reduces infractions for Interfuser, though M4PC (SGG) shows a clear trade-off with an increase in red light infractions compared to T4PC, which is discussed in a subsequent paragraph.

Both M4PC (GT) and M4PC (SGG) also reduce route timeouts compared to the Interfuser Base and T4PC models. This is accompanied by an increase in vehicle blocked infractions. Consistent with the trend observed in TCP, but more pronounced, Interfuser encounters many situations like the ones shown in Figure 5.19 where pedestrians and bicycles–considered vehicles in CARLA–get stuck in front of the ego vehicle. When this happens, M4PC prevents further motion compared to Base and T4PC models where it would slowly push forward, collide, and continue. This increase in vehicle blocked infractions across both systems is not a limitation of M4PC, but rather an indication that it prioritizes safety by halting the vehicle, instead of continuing and causing a collision. Addressing such blocked scenarios is the responsibility of the planner, not the monitor, as the monitor's role is restricted to enforcing safety via immediate correction, such as halting the vehicle, leaving the task of generating alternative, viable trajectories to the planning module. Integrating re-planning could further reduce timeouts and improve route completion.

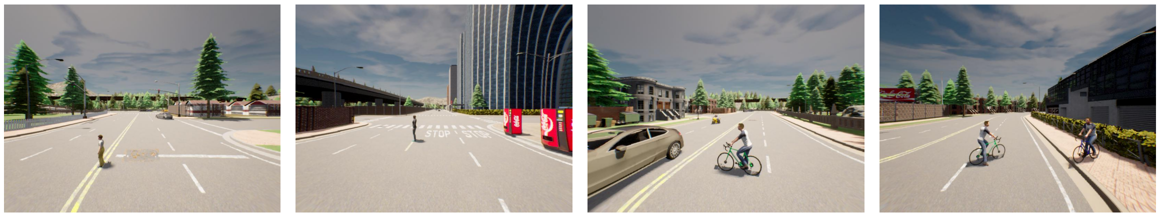


Figure 5.19: Illustrative examples for pedestrian and bicycles blocking ego vehicle.

When T4PC is combined with M4PC (GT), it outperforms the Base and T4PC baselines across all safety metrics. The T4PC + M4PC (GT) combination reduced pedestrian collisions to 0.12 (54% from Base, 33% from T4PC), vehicle collisions to 0.44 (31% from Base, 39% from T4PC), and red light infractions to 0.14 (67% from Base, 50% from T4PC). However, the average driving score is 63.24, which is above the Base but slightly below T4PC. The decline in driving score is caused by an increase in vehicle blocked infractions, which rised from 0.00 in Base and T4PC to 0.24. While M4PC successfully halts the vehicle and prevent safety violations; this stagnation reduces route completion, which in turn lowers the composite driving score.

Compared to M4PC (GT) alone, T4PC + M4PC (GT) achieves similar safety performance but yields a lower driving score. M4PC (GT) incurs 13 vehicle-blocked infractions, yet those routes achieve 42% completion on average, whereas T4PC + M4PC (GT), despite having only 12 such infractions, completes 34% of the route on average. Furthermore, M4PC (GT) attains slightly fewer infractions overall because the underlying models exhibit different failure patterns. A qualitative analysis of vehicle collision infractions revealed that Base typically collides head-on with other vehicles—cases corrected by ϕ_1 (stop if an entity is within 10m in front)—while T4PC more often collides laterally, a pattern that the property does not address. Similarly, for red light infractions, the T4PC + M4PC (GT) incurred 7 violations compared to only 2 for M4PC (GT). This increase is driven by the T4PC model's higher likelihood of failing to stop when traffic lights suddenly turn red, evidenced by five violations occurring at the exact same location across different runs where the Base model succeeded. These failures happen because ϕ_2 (if there is a yellow/red traffic light, ego should stop) does not explicitly cover abrupt signal changes, preventing effective correction in these dynamic scenarios.

The T4PC + M4PC (SGG) combination also reduces the majority of safety metrics compared to the Base and T4PC baselines. T4PC + M4PC (SGG) reduced pedestrian collisions to 0.12 (54% from Base, 33% from T4PC) and vehicle collisions to 0.56 (13% from Base, 22% from T4PC). Its red light infractions (0.32) were a 24% decrease from Base but a 14% increase over T4PC. Overall, T4PC + M4PC (SGG) achieved a driving score of 66.5, higher than T4PC + M4PC (GT), at the expense of higher collisions with vehicles and red light infractions. This inverse relationship is driven by the lower vehicle blocked frequency, 0.06 in T4PC + M4PC (SGG) compared to 0.24 in T4PC + M4PC (GT). Perception failures in M4PC (SGG) can prevent the monitor from triggering safety stops, allowing the vehicle to proceed unsafely, thereby favoring route completion over lowering collisions. We note that this trade-off is specific to Interfuser, as the properties enforced by M4PC on TCP (ϕ_7–ϕ_8) focused exclusively on stop signs and not on collision avoidance. Consequently, M4PC did not trigger similar safety halts, thereby avoiding an increase in the vehicle blocked infractions.

The increase in red light infractions observed in M4PC (SGG) (35%) and T4PC + M4PC (SGG) (14%) over Interfuser T4PC occurs despite the presence of property ϕ_2 (if there is a yellow/red light, ego vehicle should stop). This is likely due to UniAD's challenges in accurately detecting traffic lights at a distance, which can lead to missed signals and subsequent violations. This challenge is recognized by UniAD's authors in Table 6 [207], where they show that UniAD is 37% more accurate at detecting objects within 30 meters than 50 meters.

Pylot

M4PC also delivers improvements over the baseline. The ground-truth variant achieves an average driving score of 80.53, outperforming the Pylot baseline (68.96)—deemed significant by an ANOVA with $p = 0.0155$ with Bonferroni correction. This gain is accompanied by reductions in vehicle collisions by 38% (to 0.20), red light infractions by 30% (to 0.38), and stop sign infractions by 100% (to 0.00). The SGG variant also improves over the baseline with a driving score of 75.38, reducing vehicle collisions by 25% (to 0.24), and stop sign infractions by 45% (to 0.24). However, it does not reduces red light infractions despite having property ϕ_2, for the same reason explained above -

UniAD struggles at detecting objects that are far.

Summary

Overall, the results highlight three key findings. First, runtime correction via M4PC is consistently effective across different system architectures, at reducing safety-critical violations and for two out of the three ADS improving overall driving performance. Second, while ground-truth scene graphs yield the best outcomes, the SGG variant also provides infraction reductions, underscoring the practicality of our approach in realistic settings where ground-truth information is not available. Third, although our method does not directly aim to outperform T4PC—which focuses on retraining rather than runtime repair—our results demonstrate comparable average driving score, and similar or superior infraction reductions, thereby establishing M4PC as a complementary technique that can enhance conformance without retraining models. Importantly, unlike T4PC, our method does not require access to model weights or incur model retraining costs, yet it achieves similar improvements through runtime correction alone. Fourth, combining T4PC with M4PC yields mixed results. For TCP, the combination proved synergistic, achieving the highest driving scores and lowest infraction rates. In contrast, for Interfuser, the combination resulted in lower driving score and slightly lower infraction reductions due to different behavioral patterns in the underlying models that are not precisely covered by the properties. This analysis applies M4PC to diverse systems, demonstrating its capability to improve property conformance.

5.3.2.3 RQ#2: M4PC performance & effectiveness trade-off

To answer our second research question, we investigate the trade-off between M4PC's runtime performance and its corrective effectiveness. This analysis centers on the M4PC (SGG) variant for two reasons: first, it is the configuration capable of operation outside of simulation, and second, the inference latency of its perception model defines M4PC maximum operating frequency. The total inference time for M4PC (SGG) is 0.8384 seconds. This time is composed of UniAD inference (0.787 seconds), SG creation (0.05 seconds), and the property check and correction logic (0.0014 seconds). The monitor and fix portion is negligible, with the UniAD inference being the main bottleneck. This total time limits the monitor to a maximum frequency of 1.19 Hz. This 1.19 Hz frequency is low compared to the ADS, as TCP operates at 83.32 Hz (0.012 seconds) and Interfuser at 50 Hz (0.02 seconds).

In Section 5.3.2.2, M4PC (SGG) was evaluated at 2 Hz. This frequency was chosen to match the 2 Hz rate of the CARLA SGG used in the M4PC (GT), ensuring a fair comparison between the two variants by isolating the impact of scene graph quality from monitoring frequency. We note that this evaluation rate was achievable despite the model's latency because the simulator operates synchronously, pausing the simulation clock to await the completion of the monitor's computation before advancing the next step. However, as established in the previous paragraph, the 1.19 Hz frequency of M4PC (SGG) means that a 2 Hz rate is not achievable in a realistic setting. Therefore, to assess the impact of this practical constraint, we compare the performance of the 2 Hz configurations from RQ1 against a new configuration where M4PC (SGG) runs at 1 Hz. By evaluating at this rate, we can assess how a reduced monitoring frequency impacts M4PC's ability to prevent property violations. We explicitly do not evaluate at the ADS's native control frequencies (e.g., 50 Hz) because the 2 Hz configuration in Section 5.3.2.2 already serves as an optimistic upper bound. Evaluating M4PC at higher rates would simply ignore the inference latency of the SGG, contradicting the objective of this analysis which is to ground the evaluation in realistic deployment constraints.
Table 5.8 presents the results of this frequency comparison for the TCP and Interfuser baselines. In this research question, we compare the M4PC 1 Hz and 2 Hz configurations, while also contextualizing their results relative to T4PC. Consequently, we exclude Pylot from this analysis, as its architecture is incompatible with T4PC.

Treatment	Driving Score ↑	Collision Pedestrians ↓	Collision Vehicles ↓	Red Light Infraction ↓	Route Timeout ↓	Stop Sign Infraction ↓	Vehicle Blocked ↓
TCP							
Base	76.86±23.43	0.00±0.00	**0.10±0.36**	0.06±0.24	**0.00±0.00**	1.10±1.23	**0.00±0.00**
T4PC	78.23±21.76	0.00±0.00	0.12±0.33	**0.00±0.00**	**0.00±0.00**	0.94±1.13	0.02±0.14
M4PC (SGG) - 2 Hz	**79.99±27.97**	0.00±0.00	0.22±0.55	0.06±0.31	**0.00±0.00**	**0.20±0.40**	0.08±0.27
M4PC (SGG) - 1 Hz	73.00±24.34	0.00±0.00	0.18±0.39	**0.04±0.20**	0.02±0.14	0.84±1.11	0.04±0.20
Interfuser							
Base	59.41±32.28	0.26±0.49	0.64±0.88	0.42±0.67	0.06±0.24	0.00±0.00	**0.00±0.00**
T4PC	64.05±35.47	0.18±0.39	0.72±1.07	**0.28±0.50**	0.18±0.39	0.00±0.00	**0.00±0.00**
M4PC (SGG) - 2 Hz	**67.84±30.98**	**0.08±0.27**	**0.40±0.73**	0.38±0.73	**0.00±0.00**	0.00±0.00	0.14±0.35
M4PC (SGG) - 1 Hz	64.05±31.48	0.12±0.33	0.68±0.84	0.40±0.61	0.04±0.20	0.00±0.00	0.02±0.14

Table 5.8: TCP, and Interfuser results for RQ#2. Values in bold are the best.

For TCP, the 2 Hz M4PC (SGG) configuration achieves the highest driving score (79.99), an improvement over the Base (76.86) and T4PC (78.23). This is primarily due to a reduction in stop sign infractions from 1.10 in the Base and 0.94 in T4PC to 0.20. In contrast, the 1 Hz configuration results in a lower driving score (73.00) in part due to higher collisions with vehicles. As explained in RQ#1, this could be mitigated by monitoring and fixing additional collision properties. Nevertheless, the 1 Hz rate still reduces stop sign infractions to 0.84, a 24% decrease compared to Base and a 11% decrease compared to T4PC. The increase in stop sign infractions compared to the 2 Hz configuration is because the lower frequency delays the monitor's detection of an intersection, leaving little time for the corrector to intervene and make the ADS come to a stop.

For Interfuser, M4PC (SGG) also achieves the highest driving scores. In contrast, the 1 Hz (64.05) configuration gets the same score as in T4PC, and improves the average driving score compared to Base (59.41). The biggest improvement is in pedestrian collisions, which are reduced from 0.26 in Base, and from 0.18 in T4PC to 0.12 (a 54% and 33% decrease respectively), showing that even at the 1 Hz rate, M4PC (SGG) can reduce these critical infractions. When looking at vehicle collisions, the 1 Hz configuration reduces the infraction from 0.72 in T4PC to 0.68 (a 6% decrease) but it increases from 0.64 in Base (a 6% increase). However, the 2 Hz rate configuration reduces the collisions from 0.64 in Base to 0.40 (a 38% decrease), showing a trade-off between M4PC operation frequencies. This is because the lower 1 Hz frequency delays the monitor's detection of impending collisions, e.g., a car coming from the side at an intersection, leaving insufficient time for the corrector to intervene. Both monitoring rates also show reductions in red light infractions, decreasing from 0.42 in Interfuser Base to 0.38 (a 10% decrease) at 2 Hz and 0.40 (a 5% decrease) at 1 Hz. Although these values are higher than T4PC (0.28), they represent an improvement over the Base, showcasing M4PC's ability to improve property conformance during runtime. Overall, these results demonstrate that while the 2 Hz rate is more effective, the 1 Hz configuration still provides a safety benefit by improving the driving score and reducing some of the infractions.

In summary, the results for both TCP and Interfuser show a clear trade-off between monitoring frequency and corrective effectiveness. The 2 Hz rate consistently yields better driving scores and lower infraction rates than the 1 Hz rate. However, even when operating at the practical 1 Hz rate, M4PC (SGG) still demonstrates its value by reducing safety-critical infractions compared to Base and T4PC in most of the cases. The primary bottleneck is the perception model's inference time, indicating that M4PC's effectiveness is poised to improve as the perception systems get faster.

5.3.2.4 Threats to validity

Our evaluation has several limitations that should be considered when interpreting the results.

The primary threat is to external validity, as our entire evaluation is conducted within the CARLA simulator as stated in Section 5.1.2.3. While this provides a controlled and reproducible environment, the performance of M4PC may not directly transfer to real-world driving due to the well-known sim-to-real gap. Furthermore, our experiments are limited to a single environment (Town 05) and a fixed set of ten routes under clear weather conditions. The effectiveness of our approach in more diverse scenarios, such as adverse weather, different geographic locations, or more complex traffic patterns, remains an open question for future work.

Another set of threats relates to construct and internal validity. First, the performance of the M4PC (GT) variant relies on access to ground-truth scene graphs from the simulator. This represents an idealized, best-case scenario that is not achievable in practice; its results should therefore be interpreted as an upper bound on performance. Second, the effectiveness of the more practical M4PC (SGG) variant is inherently coupled with the performance of its underlying perception model, UniAD. As noted in our analysis, any limitations of the SGG in detecting certain objects (e.g., distant traffic lights) directly impact the monitor's ability to prevent corresponding violations. Finally, for our comparison with T4PC, we selected a single, randomly chosen model from the set of five they trained. While this provides a reasonable point of comparison under identical conditions, it may not be fully representative of the average performance reported in their original work.

Finally, the generalizability of our findings regarding the effectiveness of M4PC is limited by the variability observed across the tested architectures, which prevents a definitive claim of improvement. To address this, future work should expand the scope of the analysis to additional systems and test environments to further substantiate the generality of the approach and control for the variability in the experiments to robustly quantify the statistical improvement of M4PC as compared to, and in conjunction with, T4PC.

5.3.3 Summary

This section introduced M4PC, an approach designed to transition from passive monitoring to active correction of ADS behavior during deployment. This approach addresses the limitations of VLM-based SGGs, such as the high inference latency and semantic limitations SGs, which hinder real-time correction. The evaluation conducted in the CARLA simulator using three distinct ADS architectures (TCP, Interfuser, and Pylot) demonstrated that M4PC reduces safety-related infractions. It also showed M4PC's applicability as a black-box approach to different system types, including the module-based architectures like Pylot. While M4PC's performance depends on the perception model's accuracy and speed, it is expected to improve as the underlying perception technologies become faster and more precise.

5.4 Conclusion

This chapter addressed the challenge of ensuring ADS conform to safe driving properties during deployment. We tackled three interconnected challenges identified in Section 1.1.3 through a progressive series of contributions.

We first introduced SGSM for synthesizing runtime monitors directly from formal property specifications. By leveraging SGs as an intermediate representation, we demonstrated the feasibility of continuously evaluating safety properties over time within a controlled simulation environment. This established the foundation for runtime property conformance checking, moving beyond pre-deployment validation alone.

Our second contribution addressed the challenge of generating accurate SGs from real-world camera images. We developed the DriST dataset and systematically evaluated VLMs as SGGs, showing that fine-tuned VLMs can extract spatial relationships from driving scenes. Our case study demonstrated that a monitor using VLM-predicted SGs can detect safety violations, showing the feasibility of real-world monitoring without reliance on simulation data.

The third contribution transitioned from passive monitoring to active intervention. We introduced M4PC, an approach that combines an SGG based on UniAD with a correction mechanism that minimally adjusts ADS control outputs to maintain property conformance. Evaluation across three diverse ADS architectures (TCP, Interfuser, and Pylot) demonstrated consistent reductions in safety-critical violations, with the ability to operate as a black-box system requiring no model retraining or access to internal components.

Together, these three contributions provide a comprehensive solution for monitoring and enforcing safe driving properties during ADS deployment. By progressively building from simulation-based monitoring to real-world application and finally to proactive correction, this chapter advances the state of the art in ensuring ADS safety and reliability in dynamic, unpredictable environments.

Chapter 6

Conclusion

Autonomous Driving Systems are rapidly transitioning from research prototypes to commercial deployments, operating in increasingly complex real-world environments. Despite this expansion, vehicles continue to fail by exhibiting unsafe behaviors and collisions. These incidents indicate that current development and validation approaches are insufficient to identify these safety specification violations.

Improving ADS safety requires validating that these systems conform to safe driving properties—an objective currently impeded by three problems. First, the semantic disconnect between high-dimensional sensor data and high-level safe driving rules prevents the automated validation of ADS behavior. Overcoming this barrier enables the evaluation of safety rules over raw inputs. Second, current development processes often utilize datasets with insufficient coverage of safety-critical scenarios and optimize models for accuracy rather than rule adherence. Addressing this prevents ADSs from internalizing unsafe behaviors that are computationally expensive to fix after construction. Third, deployed systems lack the mechanisms to continuously monitor property conformance and intervene when violations are imminent. Solving this provides a safety layer against the stochastic nature of learning-based components and the unpredictability of real-world environments. To address these interconnected problems, this dissertation introduces SD4AS (Safe Driving for Autonomous Systems), designed to solve each problem.

First, we addressed the problem of validating systems that consume high-dimensional data against high-level safe driving rules. A primary challenge involves identifying an abstraction that retains necessary spatial information while remaining structured enough for automated evaluation. We established SGs as an abstraction layer and developed the SGL combined with LTL_f. This combination enabled the formal specification and automated evaluation of complex, temporal safe driving properties directly over sensor data, providing a mechanism to bridge the semantic gap.

Second, we focused on the problem of increasing property conformance during the development phase. This presents the challenge of ensuring that training datasets adequately cover the semantic scenarios relevant to safety properties. We introduced S^3C, an approach to automatically quantify dataset coverage. A second challenge in this phase is that simple exposure to data does not improve the model's safe behaviors. We developed T4PC, an approach that integrates safe driving properties directly into the DNN training loop. By penalizing property violations during training, T4PC provided empirical evidence that ADSs can improve their conformance to safety rules more effectively than through data exposure alone.

Third, we addressed the problem of maintaining conformance during deployment. This involves the challenge of continuously monitoring safety properties in real-time. We presented SGSM, an approach for synthesizing runtime monitors that continuously evaluate safety properties. To address the challenge of applying this in the real world without relying on simulation ground truth, we explored VLMs as SGGs. Finally, we transitioned from passive monitoring to active enforcement with M4PC, an approach capable of correcting ADS control outputs in real-time to prevent imminent violations, subject to the latency constraints of the underlying perception system.

Collectively, the contributions of this dissertation provide a unified pathway to define, build, evaluate, and enforce safe driving behaviors, advancing the state of the art in ADS safety assurance.

6.1 Broader Impacts

Autonomous Driving Systems offer the potential to improve road safety by reducing human error [9], optimizing traffic efficiency, and expanding mobility access [1]. However, while commercial services are becoming operational, achieving widespread adoption by society in complex real-world environments, requires that they operate safely and adhere to established driving rules.

By providing SD4AS to improve conformance with safe driving rules, this work contributes to the body of knowledge to enhance ADS safety. It offers a way for developers to validate that their systems comply with the driving rules, and for regulators to check that ADS abide by the law. To facilitate the adoption of these methods by the scientific community and industry practitioners, the core components of this dissertation have been disseminated through peer-reviewed publications [27, 28, 30, 31, 32]. We accompany these publications with open-source implementations and datasets [A1, A2, A3, A4, A5, A6]. These resources enable industry and researchers to scrutinize, replicate, and extend the contributions presented in this work. Ultimately, these contributions assist in paving the way for the safe integration of autonomous vehicles into society, enhancing our ability to ensure their correct operation.

While this dissertation focused on autonomous driving, we note that the core principles—using semantic graphs as an abstraction for high-dimensional data, specifying tasks via temporal logic, and enforcing conformance through runtime correction—are applicable to other autonomous domains. Systems such as warehouse robots, delivery drones, or surgical assistants can leverage similar approaches to ensure adherence to safety protocols, such as enforcing delivery constraints or maintaining safe distances in collaborative environments.

6.2 Future Work

While SD4AS offers a framework for improving property conformance, there remain open avenues for research to further enhance the scalability, expressiveness, and robustness of these techniques.

6.2.1 Bridging Sensor Inputs and Safe Driving Properties

The formalization of driving laws in Chapter 3 demonstrates the utility of SGL. However, as described in Section 3.1.3.5, the current approach uses privileged information from simulation and does not integrate perception uncertainty. The reliance on deterministic logic assumes the generated scene graph are ground truth, ignoring the confidence levels of the detected entities and relationships. Future work should explore probabilistic specification languages that can explicitly model this uncertainty. Rather than binary evaluations, a probabilistic approach would reason about the likelihood of a property holding based on the confidence scores of the underlying SGG, providing a more adaptable validation metric.

In addition to accounting for perceptual uncertainty, generating effective abstractions for complex road topologies, particularly intersections, presents another challenge. Current abstractions are either too poor, omitting information necessary to validate properties, or conversely, are overly detailed. In the latter case, to model the complex directionality of lanes, an intersection is decomposed into numerous overlapping road segments. This fragmentation complicates the evaluation of high-level semantic queries, such as determining if another vehicle shares the ego's lane as it traverses the intersection. To address this, future work should focus on developing hierarchical graph abstractions that aggregate these segments into higher-level semantic units, enabling reasoning about intersection-level dynamics.

Even when entities are correctly identified and mapped, the approach faces a limitation in tracking specific instances across time as described in Section 3.2.4. Converting graph queries into Boolean propositions causes information loss regarding object identity. For instance, if the system detects a vehicle on the left at one timestep, it cannot verify if a vehicle detected on the left in the next timestep is the same entity or a new one. This prevents the precise monitoring of rules that depend on the history of specific actors, such as right-of-way precedence at intersections. Resolving this requires the ability to distinguish between different objects in sequential frames to ensure the correct evaluation of complex dynamic scenarios. As a first step toward this goal, we published a paper that adds support for specifications that require tracking objects over time [29].

6.2.2 Increasing Property Conformance in ADS Development

Addressing the computational and structural limitations of S^{3}C described in Section 4.1.2.6 remains a priority. Currently, S^{3}C relies on graph isomorphism for clustering, a process that becomes computationally intensive as the dataset size and graph complexity grow. Future work should investigate the use of graph embeddings to approximate similarity, thereby reducing the reliance on strict isomorphism checks. Furthermore, extending the analysis to input sequences faces the challenge of interpretability. While individual scene graphs are semantically defined, a sequence of abstract class labels lacks an explicit mapping to a high-level behavior. Future research should study mapping these sequences to specific maneuvers, such as “overtaking” or “yielding”. This would enable the grouping of diverse sequences into meaningful categories, resolving sparsity issues while ensuring the results remain human-interpretable.

Complementing these structural optimizations, future work must also address how to efficiently fill the data deficiencies exposed by the analysis. Section 4.1 highlighted the ability of S^3C to quantify dataset coverage and expose specific semantic gaps, such as the absence of near-collision scenarios. However, filling these gaps currently relies on passive data collection or manual augmentation. This reliance on unguided processes results in diminishing returns in coverage improvement, as simply accumulating more data often fails to capture rare corner cases effectively. Future work should extend this approach by developing coverage-guided test generation methods that utilize the identified coverage gaps to automatically synthesize relevant scenarios using simulation or generative models. Directing the generation process toward missing semantic configurations will enable the construction of datasets that adequately cover the operational domain with greater efficiency than unguided collection.

Beyond coverage-guided data generation, the second direction addresses the limitation of T4PC regarding the expressiveness of the properties used for training described in Section 4.2.1.5. The current approach restricts the optimization process to properties defined over single input-output pairs using RFOL, which prevents the enforcement of behaviors that span multiple time steps. As noted in the limitations, this constraint hinders the model from learning more complex temporal rules, such as right-of-way negotiation, which require reasoning over sequences of inputs. Future research should investigate the integration of differentiable logic layers capable of calculating and backpropagating gradients derived from LTL_f property violations over input sequences. Enabling the loss function to account for temporal dependencies will allow the model to learn and internalize long-term safety behaviors directly during the training process.

6.2.3 Increasing Property Conformance in ADS Deployment

SGSM established the ability to synthesize a runtime monitor directly from formal property specifications. However, manually encoding every safe driving rule using SGL and LTL$_f$ format imposes a significant burden on the developer as described in Section 5.1.1.3. Future research should explore methods to automate this translation process, potentially by leveraging LLMs to synthesize formal specifications directly from natural language descriptions. Automating this step will facilitate the rapid expansion of the library of monitored properties and reduce the formal logic expertise required to implement new safe driving properties.

Once the properties are defined, the system requires an accurate representation of the environment to evaluate them. To this end, Section 5.2 demonstrated the feasibility of using VLMs to generate SGs directly from real-world camera images, enabling monitoring without privileged simulation data. Despite this capability, the generated SG currently lack important semantic details, such as road topology, and the models suffer from high inference latency that hinders real-time application as described in Section 5.2.1.3. Future work should focus on fine-tuning models to extract more environmental information while simultaneously investigating model distillation techniques to reduce inference time. Addressing these limitations will enable the deployment of VLM based SGGs that provide the comprehensive semantic detail necessary for complex property evaluation within the timing constraints of an ADS.

Moving beyond passive detection, Section 5.3 introduces M4PC, an approach to proactively correct control outputs to enforce property conformance. However, the effectiveness of this approach is bounded by the speed and accuracy of the underlying SGG as described in Section 5.3.1.4. Limited inference speeds reduce the frequency of property checks, potentially allowing violations to occur between frames, while perception errors can lead to incorrect interventions. To address these limitations, future research should investigate new SGG architectures to better capture entities and their relationships, improving accuracy while reducing inference time. Furthermore, proactively correcting for complex temporal properties, such as yielding, remains a challenge because the required action depends on the environment and system's history encoded in the monitor's state. Unlike instantaneous rules, determining the safe action requires ensuring the current output does not trigger a transition to a failure state in the monitor's automaton. Future work should develop solutions that map each monitor state to specific output constraints or utilize trajectory planning to satisfy these dynamic requirements. This will enable the system to identify the specific adjustments necessary to maintain conformance over time.

Artifacts

[A1] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B. Dwyer. *CARLA Scene Graphs*. 2025. URL: `https://github.com/less-lab-uva/carla_scene_graphs`.

[A2] Felipe Toledo, Trey Woodlief, Sebastian Elbaum, and Matthew B. Dwyer. *SGSM: Scene Graph Safety Monitoring*. 2024. URL: `https://github.com/less-lab-uva/SGSM`.

[A3] Trey Woodlief, Felipe Toledo, Matthew Dwyer, and Sebastian Elbaum. *SceneFlowLang*. 2025. URL: `https://github.com/less-lab-uva/SceneFlowLang`.

[A4] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B Dwyer. *S3C: Spatial Semantic Scene Coverage for Autonomous Vehicles*. 2024. URL: `https://github.com/less-lab-uva/s3c`.

[A5] Felipe Toledo, Trey Woodlief, Sebastian Elbaum, and Matthew B. Dwyer. *T4PC: Training Deep Neural Networks for Property Conformance*. 2025. URL: `https://github.com/less-lab-uva/T4PC`.

[A6] Felipe Toledo, Sebastian Elbaum, Divya Gopinath, Ramneet Kaur, Ravi Mangal, Corina S. Păsăreanu, Anirban Roy, and Susmit Jha. *DriST*. 2025. URL: `https://github.com/less-lab-uva/DriST`.

Bibliography

[1] Waymo. *Scaling our fleet through U.S. manufacturing.* https://waymo.com/blog/2025/05/scaling-our-fleet-through-us-manufacturing. Accessed: 2025-10-01. May 2025.

[2] TIME. *Waymo.* https://time.com/collections/time100-companies-2025/7289599/waymo/. Accessed: 2025-10-01. 2025.

[3] Rebecca Bellan. *Cruise and Waymo win robotaxi expansions in San Francisco.* Accessed on 02.07.2024. Aug. 2023. URL: https://techcrunch.com/2023/08/10/cruise-and-waymo-win-robotaxi-expansions-in-san-francisco/.

[4] Pras Subramanian. *Tesla's Cybercab robotaxi is finally here with 30K price tag — plus a surprise Robovan.* https://finance.yahoo.com/news/teslas-cybercab-robotaxi-is-finally-here-with-a-30k-price-tag–plus-a-surprise-robovan-071844079.html. 2024. (Visited on 12/01/2024).

[5] Zoox. *Zoox Vehicle.* https://zoox.com/vehicle. 2024. (Visited on 12/01/2024).

[6] Nuro Team. *Nuro Vehicle.* https://medium.com/nuro/introducing-our-next-generation-nuro-8c1c63488342. 2022. (Visited on 12/01/2024).

[7] Tesla. *Tesla Autopilot.* https://www.tesla.com/autopilot. 2024. (Visited on 12/01/2024).

[8] Waymo. *Bringing Waymo to more people, sooner.* https://waymo.com/blog/2025/08/bringing-waymo-to-more-people-sooner. Accessed: 2025-10-01. Aug. 2025.

[9] Kristofer D. Kusano, John M. Scanlon, Yin-Hsiu Chen, Timothy L. McMurry, Tilia Gode, and Trent Victor. "Comparison of Waymo Rider-Only crash rates by crash type to human benchmarks at 56.7 million miles". In: *Traffic Injury Prevention* 0.0 (2025). PMID: 40378124, pp. 1–13. DOI: 10.1080/15389588.2025.2499887. eprint: https://doi.org/10.1080/15389588.2025.2499887. URL: https://doi.org/10.1080/15389588.2025.2499887.

[10] Derek Staahl. *What happens when a Waymo is at fault in a crash?* Accessed on 11.22.2025. May 2025. URL: https://www.azfamily.com/2025/05/01/what-happens-when-waymo-is-fault-crash/.

[11] Aarian Marshall. *Uber video shows the kind of crash self-driving cars are made to avoid.* Accessed on 02.07.2024. Mar. 2018. URL: https://www.wired.com/story/uber-self-driving-crash-video-arizona/.

[12] NTS Board. *Collision between vehicle controlled by developmental automated driving system and pedestrian. Nat. Transpot. Saf. Board, Washington, DC.* Tech. rep. USA, Tech. Rep. HAR-19-03, 2019. URL: https://www.ntsb.gov/investigations/AccidentReports/Reports/HAR1903.pdf.

[13] Evan Symon. *San Francisco Cruise Robotaxi Folds Following $10 Billion In Losses.* https://californiaglobe.com/fr/san-francisco-cruise-robotaxi-folds-following-10-billion-in-losses/. Accessed: 2025-10-01. Dec. 2024.

[14] Neal E Boudette and Niraj Chokshi. "U.S. Will Investigate Tesla's Autopilot System Over Crashes With Emergency Vehicles". In: *New York Times* (Aug. 2021). Accessed on 02.07.2024. URL: https://www.nytimes.com/2021/08/16/business/tesla-autopilot-nhtsa.html.

[15] Brad Templeton. "Tesla In Taiwan Crashes Directly Into Overturned Truck, Ignores Pedestrian, With Autopilot On". In: *Forbes* (June 2020). Accessed on 02.07.2024. URL: https://www.forbes.com/sites/bradtempleton/2020/06/02/tesla-in-taiwan-crashes-directly-into-overturned-truck-ignores-pedestrian-with-autopilot-on/?sh=20a7458f58e5link.

[16] Abhirup Roy and Hyunjoo Jin. *California regulator probes crashes involving GM's Cruise robotaxis*. Accessed on 02.07.2024. Aug. 2023. URL: https://www.reuters.com/business/autos-transportation/gms-cruise-robotaxi-collides-with-fire-truck-san-francisco-2023-08-19/.

[17] Iain Thomson. *Don't tell Elon, he'd have Tesla's Robotaxis going ludicrous speed*. https://www.theregister.com/2025/09/29/california_cops_self_driving/. Accessed: 2025-10-01. Sept. 2025.

[18] AAA Newsroom. *AAA: Fear in Self-Driving Vehicles Persists*. https://newsroom.aaa.com/2025/02/aaa-fear-in-self-driving-vehicles-persists/. Accessed: 2025-10-01. Feb. 2025.

[19] Mashable. *A convincing majority of Americans remain 'afraid' to ride in self-driving vehicles*. https://mashable.com/article/how-americans-feel-self-driving-cars. Accessed: 2025-10-01. 2025.

[20] YouGov. *Americans warm to driverless cars, though skepticism remains*. https://today.yougov.com/technology/articles/51199-americans-warm-to-driverless-cars-though-skepticism-remains. Accessed: 2025-10-01. 2024.

[21] National Highway Traffic Safety Administration. *Research and Rulemaking Activities on Vehicles Equipped with Automated Driving Systems Report*. https://www.nhtsa.gov/sites/nhtsa.gov/files/2025-07/report-congress-research-rulemaking-automated-driving-systems-july-2025-tag.pdf. Accessed: 2025-10-01. July 2025.

[22] National Highway Traffic Safety Administration. *NHTSA Proposes National Program for Vehicles with Automated Driving Systems*. https://www.nhtsa.gov/press-releases/nhtsa-proposes-national-program-vehicles-automated-driving-systems. Accessed: 2025-10-01. Dec. 2024.

[23] National Highway Traffic Safety Administration. *Standing General Order on Crash Reporting*. https://www.nhtsa.gov/laws-regulations/standing-general-order-crash-reporting. Accessed: 2025-10-01. 2025.

[24] Zhuoqian Yang, Zengchang Qin, Jing Yu, and Tao Wan. "Prior Visual Relationship Reasoning For Visual Question Answering". In: *2020 IEEE International Conference on Image Processing (ICIP)*. 2020, pp. 1411–1415. DOI: 10.1109/ICIP40778.2020.9190771.

[25] Iro Armeni, Zhi-Yang He, Amir Zamir, Junyoung Gwak, Jitendra Malik, Martin Fischer, and Silvio Savarese. "3D Scene Graph: A Structure for Unified Semantics, 3D Space, and Camera". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 5663–5672. DOI: 10.1109/ICCV.2019.00576.

[26] Ue-Hwan Kim, Jin-Man Park, Taek-jin Song, and Jong-Hwan Kim. "3-D Scene Graph: A Sparse and Semantic Representation of Physical Environments for Intelligent Agents". In: *IEEE Transactions on Cybernetics* 50.12 (2020), pp. 4921–4933. DOI: 10.1109/TCYB.2019.2931042.

[27] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B. Dwyer. *Closing the Gap between Sensor Inputs and Driving Properties: A Scene Graph Generator for CARLA*. Preprint version available upon request. 2024.

[28] Felipe Toledo, Trey Woodlief, Sebastian Elbaum, and Matthew B. Dwyer. "Specifying and Monitoring Safe Driving Properties with Scene Graphs". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 15577–15584. DOI: 10.1109/ICRA57147.2024.10610973.

[29] Trey Woodlief, Felipe Toledo, Matthew Dwyer, and Sebastian Elbaum. "Scene Flow Specifications: Encoding and Monitoring Rich Temporal Safety Properties of Autonomous Systems". In: *Proc. ACM Softw. Eng.* 2.FSE (June 2025). DOI: 10.1145/3729382. URL: https://doi.org/10.1145/3729382.

[30] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B Dwyer. "S3C: Spatial Semantic Scene Coverage for Autonomous Vehicles". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24. Lisbon, Portugal: Association for Computing Machinery, 2024. ISBN: 9798400702174. DOI: 10.1145/3597503.3639178. URL: https://doi.org/10.1145/3597503.3639178.

[31] Felipe Toledo, Trey Woodlief, Sebastian Elbaum, and Matthew B Dwyer. "T4PC: Training Deep Neural Networks for Property Conformance". In: *IEEE Transactions on Software Engineering* (2025).

[32] Felipe Toledo, Sebastian Elbaum, Divya Gopinath, Ramneet Kaur, Ravi Mangal, Corina S. Păsăreanu, Anirban Roy, and Susmit Jha. "Monitoring Safety Properties for Autonomous Driving Systems with Vision-Language Models". In: *2025 IEEE Engineering Reliable Autonomous Systems (ERAS)*. 2025, pp. 1–8. DOI: 10.1109/ERAS63351.2025.11135768.

[33] Felipe Toledo, Trey Woodlief, Sebastian Elbaum, and Matthew B. Dwyer. "Correcting Autonomous Vehicle Behavior to Ensure Rule Compliance". In: *Under submission to ICRA 2026* (2025). Pre-print version available upon request.

[34] Philip Koopman and William Widen. "Breaking the Tyranny of Net Risk Metrics for Automated Vehicle Safety". In: *SSRN Electronic Journal* (Jan. 2023). DOI: 10.2139/ssrn.4634179.

[35] Yang Tang, Chaoqiang Zhao, Jianrui Wang, Chongzhen Zhang, Qiyu Sun, Wei Xing Zheng, Wenli Du, Feng Qian, and Jürgen Kurths. "Perception and Navigation in Autonomous Systems in the Era of Learning: A Survey". In: *IEEE Transactions on Neural Networks and Learning Systems* 34.12 (2023), pp. 9604–9624. DOI: 10.1109/TNNLS.2022.3167688.

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

[37] Hongsheng Li, Guangming Zhu, Liang Zhang, Youliang Jiang, Yixuan Dang, Haoran Hou, Peiyi Shen, Xia Zhao, Syed Afaq Ali Shah, and Mohammed Bennamoun. "Scene Graph Generation: A comprehensive survey". In: *Neurocomput.* 566.C (Mar. 2024). ISSN: 0925-2312. DOI: 10.1016/j.neucom.2023.127052. URL: https://doi.org/10.1016/j.neucom.2023.127052.

[38] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. "YOLOv10: Real-Time End-to-End Object Detection". In: *arXiv preprint arXiv:2405.14458* (2024).

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

[40] Bo Dai, Yuqi Zhang, and Dahua Lin. "Detecting Visual Relationships with Deep Relational Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3298–3308. DOI: `10.1109/CVPR.2017.352`.

[41] Wentong Liao, Bodo Rosenhahn, Ling Shuai, and Michael Ying Yang. "Natural Language Guided Visual Relationship Detection". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 444–453. DOI: `10.1109/CVPRW.2019.00058`.

[42] Jaewon Jung and Jongyoul Park. "Visual Relationship Detection with Language prior and Softmax". In: *2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS)*. 2018, pp. 143–148. DOI: `10.1109/IPAS.2018.8708855`.

[43] Ruichi Yu, Ang Li, Vlad I. Morariu, and Larry S. Davis. "Visual Relationship Detection with Internal and External Linguistic Knowledge Distillation". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 1068–1076. DOI: `10.1109/ICCV.2017.121`.

[44] Yikang Li, Wanli Ouyang, Xiaogang Wang, and Xiao'Ou Tang. "ViP-CNN: Visual Phrase Guided Convolutional Neural Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7244–7253. DOI: `10.1109/CVPR.2017.766`.

[45] Yikang Li, Wanli Ouyang, Bolei Zhou, Kun Wang, and Xiaogang Wang. "Scene Graph Generation from Objects, Phrases and Region Captions". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 1270–1279. DOI: `10.1109/ICCV.2017.142`.

[46] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. "Scene Graph Generation by Iterative Message Passing". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3097–3106. DOI: 10.1109/CVPR.2017.330.

[47] Hengyue Liu, Ning Yan, Masood Mortazavi, and Bir Bhanu. "Fully Convolutional Scene Graph Generation". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11541–11551. DOI: 10.1109/CVPR46437.2021.01138.

[48] Yuren Cong, Michael Ying Yang, and Bodo Rosenhahn. "RelTR: Relation Transformer for Scene Graph Generation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.9 (2023), pp. 11169–11183. DOI: 10.1109/TPAMI.2023.3268066.

[49] Rongjie Li, Songyang Zhang, and Xuming He. "SGTR: End-to-end Scene Graph Generation with Transformer". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 19486–19496.

[50] Zeeshan Hayder and Xuming He. "DSGG: Dense Relation Transformer for an End-to-end Scene Graph Generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 28317–28326.

[51] Minghan Chen, Guikun Chen, Wenguan Wang, and Yi Yang. "Hydra-SGG: Hybrid Relation Assignment for One-stage Scene Graph Generation". In: *arXiv preprint arXiv:2409.10262* (2024).

[52] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. *Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations*. 2016. arXiv: 1602.07332 [cs.CV]. URL: https://arxiv.org/abs/1602.07332.

[53] Jingkang Yang, Yi Zhe Ang, Zujin Guo, Kaiyang Zhou, Wayne Zhang, and Ziwei Liu. "Panoptic Scene Graph Generation". In: *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVII*. Tel Aviv, Israel: Springer-

Verlag, 2022, pp. 178–196. ISBN: 978-3-031-19811-3. DOI: `10.1007/978-3-031-19812-0_11`. URL: `https://doi.org/10.1007/978-3-031-19812-0_11`.

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

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

[56] Yiming Li, Xiaoshan Yang, and Changsheng Xu. “Dynamic Scene Graph Generation via Anticipatory Pre-Training”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 13874–13883.

[57] Trong-Thuan Nguyen, Pha Nguyen, Jackson David Cothren, Alper Yilmaz, Minh-Triet Tran, and Khoa Luu. “THYME: Temporal Hierarchical-Cyclic Interactivity Modeling for Video Scene Graphs in Aerial Footage”. In: *CoRR* abs/2507.09200 (2025). arXiv: `2507.09200`.

[58] Trong-Thuan Nguyen, Pha Nguyen, Jackson Cothren, Alper Yilmaz, and Khoa Luu. “HyperGLM: HyperGraph for Video Scene Graph Generation and Anticipation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2025, pp. 29150–29160.

[59] Rongjie Li, Songyang Zhang, Dahua Lin, Kai Chen, and Xuming He. “From Pixels to Graphs: Open-Vocabulary Scene Graph Generation with Vision-Language Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 28076–28086.

[60] Justin Johnson, Agrim Gupta, and Li Fei-Fei. “Image Generation from Scene Graphs”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1219–1228. DOI: `10.1109/CVPR.2018.00133`.

[61] Guibao Shen, Luozhou Wang, Jiantao Lin, Wenhang Ge, Chaozhe Zhang, Xin Tao, Yuan Zhang, Pengfei Wan, Zhongyuan Wang, Guangyong Chen, Yijun Li, and Ying-Cong Chen. *SG-Adapter: Enhancing Text-to-Image Generation with Scene Graph Guidance*. 2024. arXiv: 2405.15321 [cs.CV].

[62] Lizhao Gao, Bo Wang, and Wenmin Wang. "Image Captioning with Scene-graph Based Semantic Concepts". In: *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*. ICMLC '18. Macau, China: Association for Computing Machinery, 2018, pp. 225–229. ISBN: 9781450363532. DOI: 10.1145/3195106.3195114. URL: https://doi.org/10.1145/3195106.3195114.

[63] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Harsimrat Kaeley, Anurag Karra, and Mohammad Abdullah Al Faruque. "Roadscene2vec: A Tool for Extracting and Embedding Road Scene-Graphs". In: *Know.-Based Syst.* 242.C (Apr. 2022). ISSN: 0950-7051. DOI: 10.1016/j.knosys.2022.108245. URL: https://doi.org/10.1016/j.knosys.2022.108245.

[64] Jiachen Li, Haiming Gang, Hengbo Ma, Masayoshi Tomizuka, and Chiho Choi. "Important Object Identification with Semi-Supervised Learning for Autonomous Driving". In: *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, PA, USA: IEEE Press, 2022, pp. 2913–2919. DOI: 10.1109/ICRA46639.2022.9812234. URL: https://doi.org/10.1109/ICRA46639.2022.9812234.

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

[66] Yuchen Zhou, Yue Zhang, Zhanwei Zhao, Kaidong Zhang, and Chao Gou. "Towards Driving Scene Understanding: A Paradigm and Benchmark Dataset for Ego-centric Traffic Scene

Graph Representation". In: *IEEE Journal of Radio Frequency Identification* 6 (2022), pp. 962–967. DOI: 10.1109/JRFID.2022.3207017.

[67] Yafu Tian, Alexander Carballo, Ruifeng Li, and Kazuya Takeda. "RSG-GCN: Predicting Semantic Relationships in Urban Traffic Scene With Map Geometric Prior". In: *IEEE Open Journal of Intelligent Transportation Systems* 4 (2023), pp. 244–260. DOI: 10.1109/OJITS.2023.3260624.

[68] Boqi Chen, Kristóf Marussy, Sebastian Pilarski, Oszkár Semeráth, and Dániel Varró. "Consistent Scene Graph Generation by Constraint Optimization". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE).* 2022, pp. 1–13. DOI: 10.1145/3551349.3560433.

[69] Yongwei Li, Tao Song, and Xinkai Wu. "Robust Construction of Spatial-Temporal Scene Graph Considering Perception Failures for Autonomous Driving". In: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC).* 2023.

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

[71] Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. *Vision-Language Models for Vision Tasks: A Survey.* 2024. arXiv: 2304.00685 [cs.CV]. URL: https://arxiv.org/abs/2304.00685.

[72] Ron Mokady, Amir Hertz, and Amit H. Bermano. "ClipCap: CLIP Prefix for Image Captioning". In: *CoRR* abs/2111.09734 (2021). arXiv: 2111.09734. URL: https://arxiv.org/abs/2111.09734.

[73] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015.* Ed. by Francis R. Bach

and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 2048–2057. URL: http://proceedings.mlr.press/v37/xuc15.html.

[74] Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Devi Parikh, and Dhruv Batra. "VQA: Visual Question Answering - www.visualqa.org". In: *Int. J. Comput. Vis.* 123.1 (2017), pp. 4–31. DOI: 10.1007/S11263-016-0966-6. URL: https://doi.org/10.1007/s11263-016-0966-6.

[75] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. "Learning Transferable Visual Models From Natural Language Supervision". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8748–8763. URL: http://proceedings.mlr.press/v139/radford21a.html.

[76] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. "Grounding dino: Marrying dino with grounded pre-training for open-set object detection". In: *arXiv preprint arXiv:2303.05499* (2023).

[77] OpenAI. *GPT-5 System Card*. 2025. URL: https://cdn.openai.com/gpt-5-system-card.pdf.

[78] Gemini Team. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805.

[79] OpenAI. *GPT-5 System Card*. 2025. URL: https://cdn.openai.com/gpt-5-system-card.pdf.

[80] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. "Visual Instruction Tuning". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt,

and Sergey Levine. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html.

[81] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. "LLaMA: Open and Efficient Foundation Language Models". In: *CoRR* abs/2302.13971 (2023). DOI: 10.48550/ARXIV.2302.13971. arXiv: 2302.13971. URL: https://doi.org/10.48550/arXiv.2302.13971.

[82] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. "Improved Baselines with Visual Instruction Tuning". In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024, pp. 26286–26296. DOI: 10.1109/CVPR52733.2024.02484.

[83] Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, Zhenda Xie, Yu Wu, Kai Hu, Jiawei Wang, Yaofeng Sun, Yukun Li, Yishi Piao, Kang Guan, Aixin Liu, Xin Xie, Yuxiang You, Kai Dong, Xingkai Yu, Haowei Zhang, Liang Zhao, Yisong Wang, and Chong Ruan. *DeepSeek-VL2: Mixture-of-Experts Vision-Language Models for Advanced Multimodal Understanding.* 2024. arXiv: 2412.10302 [cs.CV]. URL: https://arxiv.org/abs/2412.10302.

[84] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. *Qwen2.5-VL Technical Report.* 2025. arXiv: 2502.13923 [cs.CV]. URL: https://arxiv.org/abs/2502.13923.

[85] Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Austin Wang, Rob Fergus, Yann LeCun, and Saining Xie. *Cambrian-1: A Fully Open, Vision-Centric Exploration of*

Multimodal LLMs. 2024. arXiv: 2406.16860 [cs.CV]. URL: https://arxiv.org/abs/2406.16860.

[86] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. "Spatialvlm: Endowing vision-language models with spatial reasoning capabilities". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 14455–14465.

[87] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. "Phi-3 technical report: A highly capable language model locally on your phone". In: *arXiv preprint arXiv:2404.14219* (2024).

[88] Abhimanyu Dubey et. al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[89] Jang Hyun Cho, Boris Ivanovic, Yulong Cao, Edward Schmerling, Yue Wang, Xinshuo Weng, Boyi Li, Yurong You, Philipp Krähenbühl, Yan Wang, and Marco Pavone. "Language-Image Models with 3D Understanding". In: *CoRR* abs/2405.03685 (2024). DOI: 10.48550/ARXIV.2405.03685. arXiv: 2405.03685. URL: https://doi.org/10.48550/arXiv.2405.03685.

[90] Tao He, Lianli Gao, Jingkuan Song, and Yuan-Fang Li. "Towards Open-Vocabulary Scene Graph Generation with Prompt-Based Finetuning". In: *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXVIII*. Ed. by Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner. Vol. 13688. Lecture Notes in Computer Science. Springer, 2022, pp. 56–73. DOI: 10.1007/978-3-031-19815-1_4. URL: https://doi.org/10.1007/978-3-031-19815-1%5C_4.

[91] Yong Zhang, Yingwei Pan, Ting Yao, Rui Huang, Tao Mei, and Chang Wen Chen. "Learning to Generate Language-Supervised and Open-Vocabulary Scene Graph Using Pre-Trained Visual-Semantic Space". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 2023, pp. 2915–2924.

DOI: 10.1109/CVPR52729.2023.00285. URL: https://doi.org/10.1109/CVPR52729.2023.00285.

[92] Rongjie Li, Songyang Zhang, Dahua Lin, Kai Chen, and Xuming He. "From Pixels to Graphs: Open-Vocabulary Scene Graph Generation with Vision-Language Models". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 2024, pp. 28076–28086. DOI: 10.1109/CVPR52733.2024.02652. URL: https://doi.org/10.1109/CVPR52733.2024.02652.

[93] Amir Pnueli. "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[94] Thomas Reinbacher, Matthias Függer, and Jörg Brauer. "Runtime verification of embedded real-time systems". In: *Formal methods in system design* 44 (2014), pp. 203–239.

[95] Srinivas Pinisetty, Partha S Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard Von Hanxleden. "Runtime enforcement of cyber-physical systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 16.5s (2017), pp. 1–25.

[96] Hengle Jiang, Sebastian Elbaum, and Carrick Detweiler. "Reducing failure rates of robotic systems though inferred invariants monitoring". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1899–1906.

[97] Giuseppe De Giacomo and Moshe Y Vardi. "Linear temporal logic and linear dynamic logic on finite traces". In: *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. Association for Computing Machinery. 2013, pp. 854–860.

[98] Shufang Zhu, Geguang Pu, and Moshe Y. Vardi. *First-Order vs. Second-Order Encodings for LTLf-to-Automata Translation*. 2019. arXiv: 1901.06108 [cs.LO]. URL: https://arxiv.org/abs/1901.06108.

[99] Francesco Fuggitti. *LTLf2DFA*. Version 1.0.0.post0. Mar. 2019. DOI: 10.5281/zenodo.3888410.

[100] Hong Zhu, Patrick A. V. Hall, and John H. R. May. “Software Unit Test Coverage and Adequacy”. In: *ACM Comput. Surv.* 29.4 (Dec. 1997), pp. 366–427. ISSN: 0360-0300. DOI: 10.1145/267580.267590. URL: https://doi.org/10.1145/267580.267590.

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

[102] Marko Ivanković, Goran Petrović, René Just, and Gordon Fraser. “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.* ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 955–963. ISBN: 9781450355728. DOI: 10.1145/3338906.3340459. URL: https://doi.org/10.1145/3338906.3340459.

[103] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. “A survey on software fault localization”. In: *IEEE Transactions on Software Engineering* 42.8 (2016), pp. 707–740.

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

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

[106] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. “Structural test coverage criteria for deep neural networks”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 18.5s (2019), pp. 1–23.

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

[108] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. “Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond”. In: *Knowledge and Information Systems* 64.12 (2022), pp. 3197–3234.

[109] T. J. Ostrand and M. J. Balcer. “The Category-Partition Method for Specifying and Generating Fuctional Tests”. In: *Commun. ACM* 31.6 (June 1988), pp. 676–686. ISSN: 0001-0782. DOI: 10.1145/62959.62964. URL: https://doi.org/10.1145/62959.62964.

[110] N. Amla and P. Ammann. “Using Z specifications in category partition testing”. In: *COMPASS ‘92 Proceedings of the Seventh Annual Conference on Computer Assurance*. 1992, pp. 3–10. DOI: 10.1109/CMPASS.1992.235766.

[111] Ajitha Rajan, Michael Whalen, Matt Staats, and Mats P. E. Heimdahl. “Requirements Coverage as an Adequacy Measure for Conformance Testing”. In: *Formal Methods and Software Engineering*. Ed. by Shaoying Liu, Tom Maibaum, and Keijiro Araki. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 86–104. ISBN: 978-3-540-88194-0.

[112] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. “A Survey on Model-Based Testing Approaches: A Systematic Review”. In: WEASELTech ’07. Atlanta, Georgia: Association for Computing Machinery, 2007, pp. 31–36. ISBN: 9781595938800. DOI: 10.1145/1353673.1353681. URL: https://doi.org/10.1145/1353673.1353681.

[113] Mark Utting, Alexander Pretschner, and Bruno Legeard. “A taxonomy of model-based testing approaches”. In: *Software Testing, Verification and Reliability* 22.5 (2012), pp. 297–312.

[114] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. “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*. 2021, pp. 79–90.

[115] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. “Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing”. In: *ACM Transactions on Software Engineering and Methodology* 32.3 (2023), pp. 1–48.

[116] Waymo LLC. *Waymo's Safety Methodologies and Safety Readiness Determinations*. Tech. rep. Oct. 2020, p. 30. URL: https://storage.googleapis.com/sdc-prod/v1/safety-report/Waymo-Safety-Methodologies-and-Readiness-Determinations.pdf.

[117] Subho S Banerjee, Saurabh Jha, James Cyriac, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. "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. 2018, pp. 586–597.

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

[119] István Majzik, Oszkár Semeráth, Csaba Hajdu, Kristóf Marussy, Zoltán Szatmári, Zoltán Micskei, András Vörös, Aren A Babikian, and Dániel Varró. "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. 2019, pp. 89–94.

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

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

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

[123] Eleni Zapridou, Ezio Bartocci, and Panagiotis Katsaros. "Runtime verification of autonomous driving systems in CARLA". In: *International Conference on Runtime Verification*. Springer. 2020, pp. 172–183.

[124] Chenxia Luo, Rui Wang, Yu Jiang, Kang Yang, Yong Guan, Xiaojuan Li, and Zhiping Shi. "Runtime verification of robots collision avoidance case study". In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2018, pp. 204–212.

[125] Aaron Lohner, Francesco Compagno, Jonathan Francis, and Alessandro Oltramari. *Enhancing Vision-Language Models with Scene Graphs for Traffic Accident Understanding*. 2024. arXiv: 2407.05910 [cs.CV]. URL: https://arxiv.org/abs/2407.05910.

[126] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Deepan Muthirayan, Pramod P. Khargonekar, and Mohammad Abdullah Al Faruque. "Spatiotemporal Scene-Graph Embedding for Autonomous Vehicle Collision Prediction". In: *IEEE Internet of Things Journal* 9.12 (2022), pp. 9379–9388. DOI: 10.1109/JIOT.2022.3141044.

[127] Alec Farid, Sushant Veer, Boris Ivanovic, Karen Leung, and Marco Pavone. "Task-relevant failure detection for trajectory predictors in autonomous vehicles". In: *Conference on Robot Learning*. PMLR. 2023, pp. 1959–1969.

[128] Huihui Wu, Deyun Lyu, Yanan Zhang, Gang Hou, Masahiko Watanabe, Jie Wang, and Weiqiang Kong. "A verification framework for behavioral safety of self-driving cars". In: *IET Intelligent Transport Systems* 16.5 (2022), pp. 630–647.

[129] Maike Schwammberger. "Distributed controllers for provably safe, live and fair autonomous car manoeuvres in urban traffic". In: 2021. URL: https://api.semanticscholar.org/CorpusID:237298372.

[130] Ankush Desai, Tommaso Dreossi, and Sanjit A Seshia. "Combining model checking and runtime verification for safe robotics". In: *International Conference on Runtime Verification*. Springer. 2017, pp. 172–189.

[131] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. "LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–12.

[132] Yang Sun, Christopher M Poskitt, Xiaodong Zhang, and Jun Sun. “REDriver: Runtime Enforcement for Autonomous Vehicles”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–12.

[133] Joseph Stamenkovich, Lakshman Maalolan, and Cameron Patterson. “Formal assurances for autonomous systems without verifying application software”. In: *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*. IEEE. 2019, pp. 60–69.

[134] André Matos Pedro, Tomás Silva, Tiago Sequeira, João Lourenço, João Costa Seco, and Carla Ferreira. “Monitoring of spatio-temporal properties with nonlinear SAT solvers”. In: *Int. J. Softw. Tools Technol. Transf.* 26.2 (Feb. 2024), pp. 169–188. ISSN: 1433-2779. DOI: 10.1007/s10009-024-00740-7. URL: https://doi.org/10.1007/s10009-024-00740-7.

[135] Christopher Morse, Lu Feng, Matthew Dwyer, and Sebastian Elbaum. “A Framework for the Unsupervised Inference of Relations Between Sensed Object Spatial Distributions and Robot Behaviors”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 901–908. DOI: 10.1109/ICRA48891.2023.10161071.

[136] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. “Reinforcement learning with temporal logic rewards”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 3834–3839. DOI: 10.1109/IROS.2017.8206234.

[137] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. “Safe reinforcement learning via shielding”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[138] Anand Balakrishnan and Jyotirmoy V. Deshmukh. “Structured Reward Shaping using Signal Temporal Logic specifications”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 3481–3486. DOI: 10.1109/IROS40897.2019.8968254.

[139] Hosein Hasanbeig, Daniel Kroening, and Alessandro Abate. “Certified reinforcement learning with logic guidance”. In: *Artificial Intelligence* 322 (2023), p. 103949. ISSN: 0004-3702. DOI:

https://doi.org/10.1016/j.artint.2023.103949. URL: https://www.sciencedirect.com/science/article/pii/S0004370223000954.

[140] Yang Sun, Christopher M. Poskitt, Kun Wang, and Jun Sun. "FixDrive: Automatically Repairing Autonomous Vehicle Driving Behaviour for $0.08 per Violation". In: *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering.* ICSE '25. Ottawa, Ontario, Canada: IEEE Press, 2025, pp. 1921–1933. ISBN: 9798331505691. DOI: 10.1109/ICSE55347.2025.00216. URL: https://doi.org/10.1109/ICSE55347.2025.00216.

[141] Mingfei Cheng, Xiaofei Xie, Renzhi Wang, Yuan Zhou, and Ming Hu. *ADReFT: Adaptive Decision Repair for Safe Autonomous Driving via Reinforcement Fine-Tuning.* 2025. arXiv: 2506.23960 [cs.LG]. URL: https://arxiv.org/abs/2506.23960.

[142] Bettina Könighofer, Julian Rudolf, Alexander Palmisano, Martin Tappler, and Roderick Bloem. "Online shielding for reinforcement learning". In: *Innovations in Systems and Software Engineering* (2022), pp. 1–16.

[143] Jacob Anderson, Georgios Fainekos, Bardh Hoxha, Hideki Okamoto, and Danil Prokhorov. "Pattern matching for perception streams". In: *International Conference on Runtime Verification.* Springer. 2023, pp. 251–270.

[144] Bo Li, Peng Qi, Bo Liu, Shuai Di, Jingen Liu, Jiquan Pei, Jinfeng Yi, and Bowen Zhou. "Trustworthy AI: From Principles to Practices". In: *ACM Comput. Surv.* 55.9 (Jan. 2023). ISSN: 0360-0300. DOI: 10.1145/3555803. URL: https://doi.org/10.1145/3555803.

[145] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Michal Walczak, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. "Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems". In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (2023), pp. 614–633. DOI: 10.1109/TKDE.2021.3079836.

[146] Anuj Karpatne, William Watkins, Jordan S. Read, and Vipin Kumar. "Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling". In: *ArXiv* abs/1710.11431 (2017). URL: `https://api.semanticscholar.org/CorpusID:3425282`.

[147] Russell Stewart and Stefano Ermon. "Label-free supervision of neural networks with physics and domain knowledge". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 2576–2582.

[148] Nikhil Muralidhar, Mohammad Raihanul Islam, Manish Marwah, Anuj Karpatne, and Naren Ramakrishnan. "Incorporating Prior Domain Knowledge into Deep Neural Networks". In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 36–45. DOI: `10.1109/BigData.2018.8621955`.

[149] Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. "Integrating Prior Knowledge into Deep Learning". In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2017, pp. 920–923. DOI: `10.1109/ICMLA.2017.00-37`.

[150] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. *A Semantic Loss Function for Deep Learning with Symbolic Knowledge*. 2018. arXiv: `1711.11157 [cs.AI]`. URL: `https://arxiv.org/abs/1711.11157`.

[151] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. "DL2: Training and Querying Neural Networks with Logic". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 1931–1941. URL: `https://proceedings.mlr.press/v97/fischer19a.html`.

[152] Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. "Semantic strengthening of neuro-symbolic learning". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 10252–10261.

[153] Xiaodan Liang, Zhiting hu, Hao Zhang, Liang Lin, and Eric P. Xing. "Symbolic graph reasoning meets convolutions". In: *Proceedings of the 32nd International Conference on Neural

Information Processing Systems. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 1858–1868.

[154] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F. Stewart, and Jimeng Sun. "GRAM: Graph-based Attention Model for Healthcare Representation Learning". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 787–795. ISBN: 9781450348874. DOI: 10.1145/3097983.3098126. URL: https://doi.org/10.1145/3097983.3098126.

[155] Hao Zhou, Tom Young, Minlie Huang, Haizhou Zhao, Jingfang Xu, and Xiaoyan Zhu. "Commonsense knowledge aware conversation generation with graph attention". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI'18. Stockholm, Sweden: AAAI Press, 2018, pp. 4623–4629. ISBN: 9780999241127.

[156] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence W. K. Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. "Lagrangian Duality for Constrained Deep Learning". In: *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*. Ed. by Yuxiao Dong, Georgiana Ifrim, Dunja Mladenić, Craig Saunders, and Sofie Van Hoecke. Cham: Springer International Publishing, 2021, pp. 118–135. ISBN: 978-3-030-67670-4.

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

[158] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. "End-to-End Autonomous Driving: Challenges and Frontiers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.12 (2024), pp. 10164–10183. DOI: 10.1109/TPAMI.2024.3435937.

[159] Trey Woodlief, Felipe Toledo, Sebastian Elbaum, and Matthew B. Dwyer. "The SGSM framework: Enabling the specification and monitor synthesis of safe driving properties through

scene graphs". In: *Science of Computer Programming* 242 (2025), p. 103252. ISSN: 0167-6423. DOI: `https://doi.org/10.1016/j.scico.2024.103252`. URL: `https://www.sciencedirect.com/science/article/pii/S0167642324001758`.

[160] 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*. Springer. 1995, pp. 704–720.

[161] *Virginia Code Title 46.2 Chapter 8 - Motor Vehicles, Regulation of Traffic.*

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

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

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

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

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

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

[168] Kelvin Wong, Yanlei Gu, and Shunsuke Kamijo. "Mapping for autonomous driving: Opportunities and challenges". In: *IEEE Intelligent Transportation Systems Magazine* 13.1 (2020), pp. 91–106.

[169] Umar Ozgunalp, Rui Fan, Xiao Ai, and Naim Dahnoun. "Multiple lane detection algorithm based on novel dense vanishing point estimation". In: *IEEE Transactions on Intelligent Transportation Systems* 18.3 (2016), pp. 621–632.

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

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

[172] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C Briand. "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. 2020, pp. 85–95.

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

[174] 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: *CVPR*. 2020.

[175] Sully Chen. *driving-datasets*. 2017. URL: `https://github.com/SullyChen/driving-datasets`.

[176] Udacity. *self-driving-car*. 2016. URL: `https://github.com/udacity/self-driving-car`.

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

[178] Virginia Department of Motor Vehicles. *Virginia Driver's Manual.* URL: `https://transactions-t.dmv.virginia.gov/webdoc/pdf/dmv39.pdf` (visited on 10/02/2024).

[179] Trey Woodlief, Sebastian Elbaum, and Kevin Sullivan. "Semantic image fuzzing of AI perception systems". In: *Proceedings of the 44th International Conference on Software Engineering.* ICSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 1958–1969. ISBN: 9781450392211. DOI: `10.1145/3510003.3510212`. URL: `https://doi.org/10.1145/3510003.3510212`.

[180] Daniel Jackson. "Alloy: a lightweight object modelling notation". In: *ACM Transactions on software engineering and methodology (TOSEM)* 11.2 (2002), pp. 256–290.

[181] Penghao Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. "Trajectory-guided control prediction for end-to-end autonomous driving: a simple yet strong baseline". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems.* NIPS '22. New Orleans, LA, USA: Curran Associates Inc., 2024. ISBN: 9781713871088.

[182] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. "Safety-enhanced autonomous driving using interpretable sensor fusion transformer". In: *Conference on Robot Learning.* PMLR. 2023, pp. 726–737.

[183] Dian Chen and Philipp Krähenbühl. "Learning from All Vehicles". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* 2022, pp. 17201–17210. DOI: `10.1109/CVPR52688.2022.01671`.

[184] Vasileios Arampatzakis et al. "Monocular Depth Estimation: A Thorough Review". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.4 (2024), pp. 2396–2414. DOI: `10.1109/TPAMI.2023.3330944`.

[185] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[186] *TCP Training Configuration.* `https://github.com/OpenDriveLab/TCP/blob/main/TCP/config.py`. Accessed: 2024-07-19.

[187] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. “Pure-past linear temporal and dynamic logic on finite traces”. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI’20. Yokohama, Yokohama, Japan, 2021. ISBN: 9780999241165.

[188] Tomer Toledo and David Zohar. “Modeling duration of lane changes”. In: *Transportation Research Record* 1999.1 (2007), pp. 71–78.

[189] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[190] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[191] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[192] Dian Chen and Philipp Krähenbühl. “Learning from all vehicles”. In: *CVPR*. 2022.

[193] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[194] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[195] Vung Pham, Chau Pham, and Tommy Dang. "Road Damage Detection and Classification with Detectron2 and Faster R-CNN". In: *CoRR* abs/2010.15021 (2020). arXiv: 2010.15021. URL: https://arxiv.org/abs/2010.15021.

[196] Glenn Jocher and Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[197] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. "Yolo-world: Real-time open-vocabulary object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 16901–16911.

[198] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. "Object Detection in 20 Years: A Survey". In: *Proceedings of the IEEE* 111.3 (2023), pp. 257–276. DOI: 10.1109/JPROC.2023.3238524.

[199] OpenAI. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[200] Arnav Vaibhav Malawade, Shih-Yuan Yu, Brandon Hsu, Harsimrat Kaeley, Anurag Karra, and Mohammad Abdullah Al Faruque. "roadscene2vec: A tool for extracting and embedding road scene-graphs". In: *Knowl. Based Syst.* 242 (2022), p. 108245. DOI: 10.1016/J.KNOSYS.2022.108245. URL: https://doi.org/10.1016/j.knosys.2022.108245.

[201] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. *PaliGemma: A versatile 3B VLM for transfer*. 2024. arXiv: 2407.07726 [cs.CV]. URL: https://arxiv.org/abs/2407.07726.

[202] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. *Sigmoid Loss for Language Image Pre-Training*. 2023. arXiv: `2303.15343 [cs.CV]`. URL: `https://arxiv.org/abs/2303.15343`.

[203] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. “Lora: Low-rank adaptation of large language models.” In: *ICLR* 1.2 (2022), p. 3.

[204] Gemma Team et al. *Gemma: Open Models Based on Gemini Research and Technology*. 2024. arXiv: `2403.08295 [cs.CL]`. URL: `https://arxiv.org/abs/2403.08295`.

[205] Marah Abdin et al. *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. 2024. arXiv: `2404.14219 [cs.CL]`. URL: `https://arxiv.org/abs/2404.14219`.

[206] Julian Lorenz, Robin Schön, Katja Ludwig, and Rainer Lienhart. “A Review and Efficient Implementation of Scene Graph Generation Metrics”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024 - Workshops, Seattle, WA, USA, June 17-18, 2024*. IEEE, 2024, pp. 2567–2575. DOI: `10.1109/CVPRW63382.2024.00263`. URL: `https://doi.org/10.1109/CVPRW63382.2024.00263`.

[207] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. “Planning-oriented Autonomous Driving”. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 17853–17862. DOI: `10.1109/CVPR52729.2023.01712`.

[208] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Matthew A Wright, Joseph E Gonzalez, and Ion Stoica. “Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8806–8813.

[209] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. “Autoware on board: Enabling autonomous vehicles with embedded systems”. In: *Proceedings of the 9th*

ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS). 2018, pp. 287–296.

[210] Baidu Apollo Team. *Apollo: Open Source Autonomous Driving*. https://github.com/ApolloAuto/apollo. 2017. (Visited on 12/01/2024).

Appendix

A Suplemental Details on DriST

The following table shows the Precision (P), Recall (R), and F1 for each model, query mode (QM) and dataset. The LlaVA 1.5 fine-tuned (ft) and LoRA (L) are the best performing models across the different query modes, showing the effectiveness of the DriST dataset, as a mean of enhancing VLMs capabilities to capture spatial relationships for driving scenarios. By taking a closer look at the precision and recall values, we can see that models like LlaVA 1.5, PaliGemma, and SpaceLlaVA, in query mode 4, achieve 100% recall; however, their precision is very low. This is because these models answer 'yes' to all yes/no questions, ensuring no triplet is missed but resulting in high imprecision.

Table A.1: Performance of different VLMs using the 4 different query modes.

Model	**QM**	**Time**	**Total**			**Kitti**			**Waymo**			**nuScenes**		
			R	**P**	**F1**	**R**	**P**	**F1**	**R**	**P**	**F1**	**R**	**P**	**F1**
C-Llama3	1	15.29	0.44	0.15	0.19	0.44	0.11	0.15	0.42	0.15	0.18	0.46	0.20	0.23
C-Phi3	1	9.85	0.57	0.19	0.26	0.65	0.23	0.31	0.55	0.20	0.26	0.49	0.16	0.21
GPT-4-T	1	5.89	0.42	0.60	0.45	0.38	0.68	0.45	0.35	0.54	0.37	0.53	0.57	0.53
L1.5	1	3.95	0.44	0.36	0.36	0.50	0.47	0.44	0.40	0.39	0.35	0.42	0.24	0.28
L1.5-FT	1	2.57	0.63	0.76	**0.66**	0.69	0.83	0.72	0.61	0.81	**0.67**	0.59	0.64	**0.59**
L1.5-L	1	2.58	0.62	0.74	0.65	0.68	0.85	**0.73**	0.62	0.80	0.66	0.56	0.59	0.55
L1.6-Mis	1	3.15	0.36	0.46	0.36	0.32	0.51	0.35	0.37	0.49	0.38	0.38	0.38	0.35
L1.6-Vic	1	3.65	0.28	0.27	0.25	0.29	0.27	0.26	0.30	0.29	0.27	0.24	0.23	0.23
PaliGemma	1	1.02	0.29	0.52	0.33	0.32	0.62	0.38	0.26	0.56	0.32	0.30	0.37	0.30
RS2V	1	0.05	0.40	0.24	0.27	0.00	0.00	0.00	0.53	0.26	0.31	0.66	0.47	0.51
SpaceLlaVA	1	11.16	0.42	0.29	0.29	0.51	0.40	0.39	0.37	0.23	0.24	0.37	0.23	0.25
C-Llama3	2	8.71	0.66	0.53	0.54	0.67	0.56	0.55	0.64	0.56	0.56	0.67	0.45	0.51
C-Phi3	2	8.20	0.64	0.51	0.52	0.63	0.53	0.51	0.64	0.57	0.56	0.65	0.44	0.49
GPT-4-T	2	108.81	0.71	0.34	0.42	0.80	0.37	0.46	0.68	0.37	0.44	0.64	0.30	0.37
L1.5	2	5.13	0.46	0.52	0.45	0.45	0.56	0.44	0.43	0.57	0.46	0.50	0.44	0.44
L1.5-FT	2	4.81	0.73	0.78	**0.74**	0.84	0.87	**0.84**	0.73	0.80	**0.74**	0.63	0.69	**0.64**
L1.5-L	2	4.84	0.64	0.75	0.67	0.65	0.81	0.69	0.66	0.79	0.69	0.62	0.66	0.61
L1.6-Mis	2	8.93	0.58	0.52	0.50	0.57	0.51	0.47	0.53	0.59	0.52	0.63	0.45	0.49
L1.6-Vic	2	8.25	0.46	0.51	0.45	0.42	0.36	0.35	0.44	0.62	0.48	0.51	0.56	0.50
PaliGemma	2	1.69	0.69	0.19	0.27	0.75	0.22	0.31	0.65	0.24	0.32	0.68	0.12	0.20
SpaceLlaVA	2	14.44	0.54	0.41	0.42	0.59	0.44	0.44	0.51	0.51	0.47	0.53	0.28	0.34
C-Llama3	3	4.59	0.47	0.47	0.47	0.45	0.45	0.45	0.40	0.40	0.40	0.55	0.55	0.55
C-Phi3	3	4.01	0.45	0.45	0.45	0.42	0.42	0.42	0.37	0.37	0.37	0.56	0.56	0.56
GPT-4-T	3	27.63	0.52	0.52	0.52	0.49	0.49	0.49	0.38	0.38	0.38	0.71	0.71	0.71
L1.5	3	9.58	0.58	0.37	0.42	0.60	0.34	0.40	0.53	0.31	0.35	0.59	0.47	0.50
L1.5-FT	3	4.30	0.89	0.89	0.89	0.87	0.87	0.87	0.90	0.90	**0.90**	0.88	0.88	0.88
L1.5-L	3	4.35	0.90	0.90	**0.90**	0.89	0.89	**0.89**	0.90	0.90	0.90	0.92	0.92	**0.92**
L1.6-Mis	3	5.31	0.45	0.45	0.45	0.44	0.44	0.44	0.39	0.39	0.39	0.53	0.53	0.53
L1.6-Vic	3	5.60	0.39	0.38	0.38	0.29	0.29	0.29	0.38	0.33	0.34	0.50	0.51	0.50
PaliGemma	3	2.82	0.39	0.40	0.39	0.34	0.36	0.35	0.36	0.37	0.36	0.47	0.48	0.47
SpaceLlaVA	3	9.39	0.31	0.29	0.30	0.27	0.25	0.26	0.32	0.30	0.30	0.33	0.33	0.33
C-Llama3	4	10.20	0.53	0.55	0.50	0.75	0.59	0.63	0.24	0.40	0.27	0.60	0.65	0.59
C-Phi3	4	9.19	0.21	0.39	0.25	0.12	0.40	0.17	0.18	0.25	0.19	0.34	0.53	0.37
GPT-4-T	4	169.72	0.64	0.63	0.61	0.70	0.53	0.59	0.40	0.59	0.46	0.82	0.77	0.78
L1.5	4	6.40	1.00	0.42	0.56	1.00	0.33	0.50	1.00	0.43	0.57	1.00	0.49	0.61
L1.5-FT	4	5.42	0.93	0.93	**0.93**	0.93	0.93	**0.93**	0.93	0.92	**0.93**	0.93	0.93	**0.93**
L1.5-L	4	5.44	0.76	0.88	0.81	0.72	0.88	0.78	0.83	0.87	0.84	0.73	0.89	0.79
L1.6-Mis	4	26.91	0.76	0.43	0.52	0.67	0.37	0.45	0.66	0.44	0.51	0.96	0.49	0.61
L1.6-Vic	4	9.72	0.84	0.45	0.55	0.93	0.35	0.50	0.72	0.48	0.54	0.87	0.53	0.63
PaliGemma	4	2.30	1.00	0.42	0.56	1.00	0.33	0.50	1.00	0.43	0.57	1.00	0.49	0.61
SpaceLlaVA	4	28.72	1.00	0.42	0.56	1.00	0.33	0.50	1.00	0.43	0.57	1.00	0.49	0.61