Correcting Autonomous Vehicle Behavior to Ensure Rule Compliance

Felipe Toledo[1], Trey Woodlief[2], Sebastian Elbaum[1], and Matthew B. Dwyer[1]

***Abstract*— As autonomous vehicles (AVs) continue to gain prominence in public life, the cost of their failures becomes increasingly drastic, endangering human life. Such failures arise from AVs' inability to meet their safety specifications in the field. Recent works have aimed to improve AVs' compliance with their safety specification through improved training and runtime enforcement. However, these methods are limited, requiring access to system internals or relying on narrow assumptions, which reduces their generality. In this work, we propose a different paradigm, Monitoring for Property Compliance (M4PC), which independently evaluates the system's compliance with the specification. The approach operates in two steps. First, it leverages scene graph abstractions and a specialized graph generator to map sensor data to driving rule preconditions to determine if an intervention is needed. Second, to correct an erroneous system output, M4PC defines a safe region within the control space defined by all relevant postconditions and minimally alters the system's output to ensure it remains within this safe region, thereby preventing property violations. We apply M4PC to improve the specification compliance of three state-of-the-art autonomous vehicles with varying architectures in the CARLA simulator. Our current implementation can improve a baseline system, while our most optimized implementation outperforms state-of-the-art techniques that require system access.**

I. Introduction

Autonomous vehicle (AV) companies are rapidly expanding their footprint, deploying more vehicles in more jurisdictions [1]. This growth makes it crucial to ensure these systems operate safely and consistently adhere to driving properties, such as traffic laws of the jurisdictions in which they operate. For example, a vehicle's failure to completely stop at a stop sign (as seen in Figure 1) or when a pedestrian is crossing can have catastrophic consequences [2], [3].

Ensuring that AVs comply with critical driving properties presents a significant challenge. The rules themselves can be subtle and context-dependent, and the input space, particularly from high-fidelity sensors like cameras and LiDAR, is enormous. Furthermore, a significant disconnect exists between the low-level sensor data and the high-level driving rules that requires integrating these different abstraction levels. Scene graphs offer a principled abstraction for bridging this gap, encoding objects, lanes, and their semantic relationships in a structured representation that maps naturally onto the preconditions of driving rules [4] (Chapter 3.1). This problem is compounded by the fact that modern AVs increasingly rely on opaque, learned components, such as deep neural networks, where the root causes of misbehavior are difficult to uncover, which is further exacerbated when these components interact with other heterogeneous AV components.

Existing approaches to assist AVs to comply with driving rules can be broadly categorized into three groups. The first category, and the one that has earned the most attention, focuses on generating testing scenarios to verify compliance during development [6], [7], [8], [9], [10]. They can generate a large number of scenarios aiming to expose problematic AV behavior, but are unable to guarantee or even estimate compliance with regards to a set of driving properties. The second category involves monitoring to detect compliance violations after a system has been deployed [11], [12], acting as a safety net for violations that escape testing.

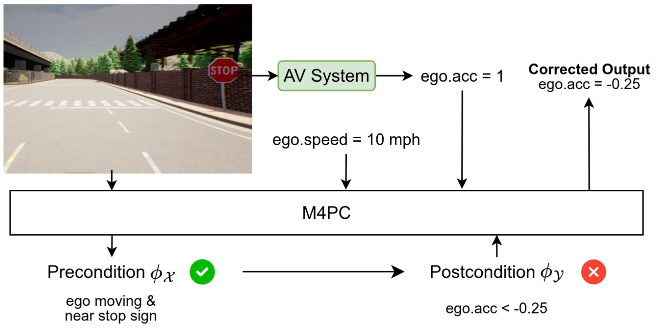

Fig. 1: Consider a driving property: when a stop sign is near and the vehicle is still moving, it must slow down. Given sensor inputs (image and velocity), M4PC checks if the property's precondition is met (e.g., a stop sign is near) and then if the vehicle's output satisfies the postcondition (e.g., negative acceleration). If not, a minimal correction is applied.

The third category consists of techniques aiming to enforce property compliance. Within this category, we distinguish between two classes of techniques: those that attempt to build compliance by construction as part of the development and training process, such as T4PC [5], and those that enforce compliance at runtime, such as `REDriver` [13]. These techniques are complementary. Building compliance into the training process is always beneficial as it reduces the need for applying corrections at runtime. However, compliance by construction does not guarantee perfect behavior; runtime correction provides an additional layer of assurance in the field. The problem with both techniques is that they make strong assumptions about the target system, requiring either a homogeneous, differentiable, machine-learned architecture [5] or a modular architecture with components that satisfy a prescribed interface and a common world representation over which to detect and correct violations [13]. These requirements limit the generalizability of these techniques.

In this work, we propose a novel and more general approach for enforcing property compliance, Monitoring for Property Compliance (M4PC). This approach supports at least as many driving rules as prior work (see sample here in Table I) but without relying

[1] University of Virginia, Virginia, USA{ft8bn, selbaum, matthewbdwyer}@virginia.edu [2] William & Mary, Virginia, USA woodlief@wm.edu

TABLE I: Table adapted from Toledo et al. [5]. Properties with preconditions ($\phi_{\mathcal{X}}$) in RFOL over SGs and postconditions ($\phi_{\mathcal{Y}}$) as AV's output constraints.

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.	$\mathit{ego.dist}(\leq,10)\cap\mathit{ego.on.\hat{}on}\cap\mathit{ego.infront}\neq\emptyset$	-1.00 $\leq$ acc $\leq$ -0.25
ϕ_2 (46.2-833)	If ego lane traffic light is red/yellow, then ego should stop.	$\mathit{ego.controlled.T_{light}.color}\in\{\mathit{red,yellow}\}$	-1.00 $\leq$ 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.	$\mathit{ego.controlled.stopsign}=\emptyset\wedge$ $\mathit{ego.controlled.T_{light}.color}\notin\{\mathit{red,yellow}\}\wedge$ $\mathit{ego.dist}(\leq,25)\cap\mathit{ego.on.\hat{}on}\cap\mathit{ego.infront}=\emptyset$	0.25 $\leq$ acc $\leq$ 1.00
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.	$\mathit{ego.controlled.T_{light}.color}=\mathit{green}\wedge$ $\mathit{ego.dist}(\leq,10)\cap\mathit{ego.on.\hat{}on}\cap\mathit{ego.infront}=\emptyset$	0.25 $\leq$ acc $\leq$ 1.00
ϕ_5 (46.2-821)	If ego is moving and there is a stop sign affecting it within 10m, then ego should stop.	$\mathit{ego.speed}\geq 0.1\wedge$ $\mathit{ego.dist}(\leq,10)\cap\mathit{ego.controlled.stopsign}\neq\emptyset$	-1.00 $\leq$ acc $\leq$ -1.00
ϕ_6 (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.	$\mathit{ego.speed}<0.1\wedge$ $\mathit{ego.dist}(\leq,10)\cap\mathit{ego.controlled.stopsign}\neq\emptyset\wedge$ $\mathit{ego.dist}(\leq,7)\cap\mathit{ego.on.\hat{}on}\cap\mathit{ego.infront}=\emptyset$	0.75 $\leq$ acc $\leq$ 0.75

on particular architectures, interfaces, or access to the system internals, significantly increasing generality. Instead, M4PC only requires the raw sensor inputs consumed by the system and its control outputs. As exemplified in Figure 1, M4PC consumes the camera image and the speedometer data of the vehicle. When M4PC detects that a precondition of ϕ_5 in Table I is satisfied—the ego vehicle is moving and has a stop sign-controlled intersection within 10 meters—and the postcondition is not satisfied—ego vehicle acceleration is not negative—it will correct the AV System's output, in this case correcting a positive acceleration of 1 to a negative acceleration of -0.25, such that the ego vehicle's actuated acceleration complies with the safe driving property. At no point does the approach require access to the system internals.

To achieve this, M4PC leverages two key mechanisms to apply corrections. First, to determine if a correction is needed, it uses scene graph abstractions to map sensor data to driving rule preconditions. Our implementation integrates state-of-the-art models that predict 3D bounding boxes and lane segmentation into a bird's-eye-view (BEV) representation. From this, it extracts various semantic relationships between objects in the scene, which provides a richer and more accurate assessment of whether a property applies to the current situation. Second, if a property's precondition is satisfied, our approach checks the system's output to ensure it meets the corresponding postcondition. If the output is correct, the system proceeds as intended. If not, M4PC applies a minimal corrective action to satisfy the property. Specifically, the property's postcondition defines a "safe" region within the control space, and our correction mechanism minimally alters the system's output to keep it within this safe region, thereby preventing property violations.

Our evaluation on three driving systems in the CARLA simulator, using six distinct driving rules, and comparing M4PC against baseline systems and those optimized for compliance during training, shows that: 1) M4PC has the potential to outperform all existing techniques when the scene graph generator is accurate, 2) with our implementation of the scene graph generator, M4PC is better than all baselines and at least comparable to the system optimized for compliance during training, and 3) M4PC is more general and applicable to all systems, including one with multiple heterogeneous and nondifferentiable components.

TABLE II: Comparison of SGG for Autonomous Driving. ✗ indicates no support, ● indicates partial, inaccurate, or insufficient support, and ✓ indicates sufficient support.

Methods	Nodes		Relationships		
	Lanes	Road Entities	Entity to Entity	Entity to Road	Traffic Infrastructure
RS2V [17]	●	✓	✓	●	✗
DriST [18]	✗	✓	✓	✗	✗
SGG [19], [20], [21]	✗	✓	●	●	✗
Ours	✓	✓	✓	✓	✓

II. Related Work and Background

We briefly survey related work on scene graphs, their use in specifying AV safe driving properties, and prior methods for improving property compliance.

A. Scene Graphs

Scene Graphs (SGs) are an abstract, parameterizable data structure for representing the physical world which have been widely used in varied fields from general scene understanding [14], [15] to representing surgical robots [16] to the roadway [17], [5], [11]. In the traffic context specifically, for example, the SG represents the world by capturing physical and logical objects in the nodes of the graph, e.g., the other road vehicles or the lanes in the road, and their physical and semantic relationships in the edges of the graph, e.g., one edge capturing that the stop sign is to the right of the vehicle and another edge capturing that the stop sign controls the traffic in that lane. Scene graphs can be further enriched with attributes for nodes and edges.

To leverage such data abstractions, we need Scene Graph Generators (SGGs). However, inaccuracies and imprecision in prior SGGs have limited their real-world applicability. A qualitative comparison of popular SGG methods is presented in Table II to highlight the strengths and limitations of existing approaches. Each SGG is judged based on its support for different elements necessary for AV specification monitoring, and thus M4PC. RoadScene2Vec (RS2V) [17] produces scene graphs tailored to driving tasks but assumes a fixed three-lane road structure, which limits its applicability to diverse road conditions and affects the accuracy of entity-to-lane relationships when the assumption does not hold. DriST [18] employs Visual Language Models (VLMs) to extract scene graphs from real images, but

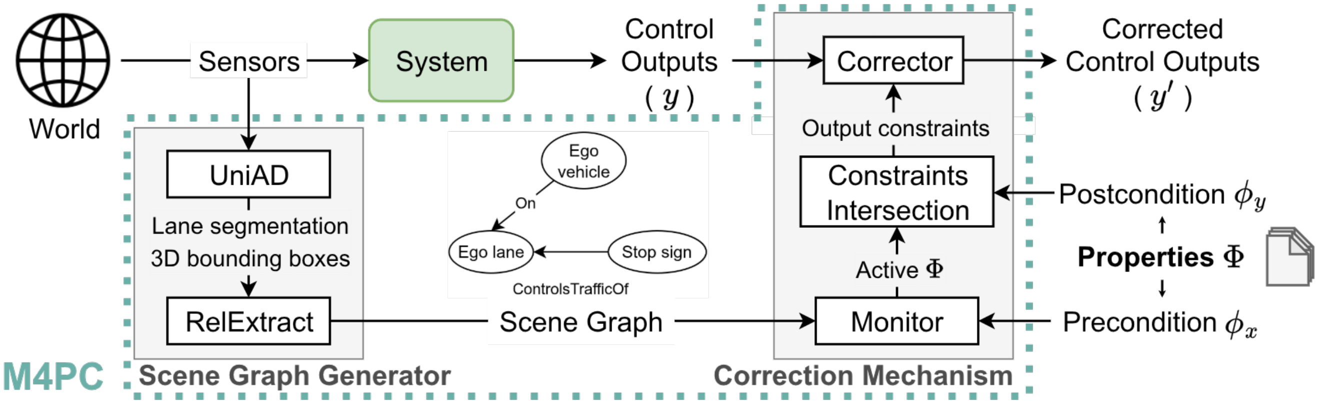

Fig. 2: M4PC Approach Diagram.

it does not detect lanes or their relationships with other entities. Generic SGGs [19], [20], [21], trained on general-purpose or indoor datasets, lack support for lane and traffic infrastructure, and do not capture driving-specific relationships between entities.

This context has forced researchers to either build graphs from ground-truth data from simulations, sacrificing realism for accuracy [5], [11], [12], [22], [23], or use simplified abstractions that miss critical elements, sacrificing accuracy and generality [17], [18]. To address this, in this work, we present an SGG that integrates multiple perception models and relational reasoning to achieve the richness and accuracy needed to correct common driving violations.

B. Relational Specifications

The rich structure of SGs enables the precise encoding of relevant safety properties through specialized logics, e.g., Relational First-Order Logic (RFOL). Prior work has shown the utility of such structured logic to express complex properties about the world as abstracted through the scene graph to enable runtime monitoring [11], [12] and dataset labeling to improve property compliance at training time [5]. We leverage this capability to specify and evaluate properties over the SG. As we compare against the technique as a baseline later, we adapt the specifications studied for T4PC [5] in our analysis. Table I shows the properties that we target in Section V in applying M4PC to existing AV systems. The first and second column show the property number, the section of the Virginia Driving Code [24] it targets, and a short English description. The third column shows the preconditions, $\phi_{\mathcal{X}}$, described using RFOL. Finally, the fourth column shows the postcondition, $\phi_{\mathcal{Y}}$, as an interval constraint on the control output.

C. Improving Property Compliance

As introduced in Section I, prior work has aimed to improve AV property compliance; two particularly relevant works are T4PC [5] and REDriver [13].

Training for Property Conformance (T4PC) aims to improve compliance by biasing the training process toward property satisfaction [5]. To do so, T4PC preprocesses the training data to generate SGs for each input and uses the SG to determine whether the property's precondition is active for that input. If so, then the loss function is augmented to penalize postcondition violations. Crucially, T4PC cannot provide runtime compliance guarantees because the training process may not generalize to unseen data, leading to field violations. This is a fundamental limitation of training-based approaches. Further, T4PC's training-based approach is only applicable to machine-learned components, precluding its application to alternative system implementations. Our experiments in Section V demonstrate M4PC's ability to improve compliance of diverse system implementations. We also remark that T4PC performed their experiment using an SGG that used ground-truth simulator state [22]. This represents a threat to the external validity of the findings. In this work, we use a sensor-based SGG that corrects AV behavior at runtime, addressing both of these limitations.

REDriver enforces compliance at runtime by interacting with the AV's internal perception, planning, and control modules to alter its behavior [13]. Unlike T4PC, this approach enables active correction of AV behavior to improve property compliance. However, this approach is tightly coupled with the AV implementation: the properties' preconditions must be specified over primitives already present in the AV's perception output, and the corrective action applied to meet the postcondition is applied at multiple levels within the AV decision stack. This coupling limits application of REDriver over diverse AVs and diverse properties. To address these limitations, M4PC operates independent of the implementation of the AV, taking as input the initial sensor data and altering only the final control outputs of the AV.

III. Approach

Our proposed approach introduces a monitoring and correction framework for autonomous systems, M4PC, which ensures robust operation by specifying safety properties over SGs, leveraging a novel SGG to evaluate property preconditions at runtime, efficiently performing in situ corrections, ensuring that the control outputs produced by the system abide by the postcondition. The pipeline, shown in Figure 2, consists of three major components: (i) Property Specification, (ii) SGG from sensor inputs, and (iii) Correction Mechanism for runtime correction. Together, these components enable the system to detect and mitigate potential failures before violating a driving rule.

A. Property Specification

As shown in Table I, properties enforced by M4PC are expressed as a precondition and postcondition pair, $\phi_{\mathcal{X}} \implies \phi_{\mathcal{Y}}$. The precondition is defined in RFOL and can be evaluated directly over the scene graph (e.g., "no pedestrian is within 5 meters of the ego vehicle's planned trajectory"), while the postcondition defines the allowable system outputs, as an interval constraint on the output domain, to satisfy the rule. M4PC takes as input a set of properties to enforce; the set of properties must be internally consistent. Given the universe of possible SGs, D, control outputs, O, and properties, Φ, then $\forall x \in D, a, b \in \Phi, a \neq b, \exists y \in O : \phi^a_{\mathcal{X}}(x) \wedge \phi^b_{\mathcal{X}}(x) \implies \phi^a_{\mathcal{Y}}(y) \wedge \phi^b_{\mathcal{Y}}(y)$. That is, for any SG that satisfies the precondition of two different properties, there must exist a control output satisfying both postconditions; otherwise, it is not possible for any implementation to meet such a specification. M4PC relies on the developer to certify that the properties are consistent. However, we can automatically check for consistency within a bounded scope by evaluating co-satisfaction of preconditions over a dataset and verifying postcondition consistency as in T4PC [5].

B. 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** [25], a foundational model for autonomous driving that performs lane segmentation and 3D object detection. It takes in multi-camera images and generates 3D bounding boxes for objects (e.g., vehicles, pedestrians, cyclists, and infrastructure) and segmentation of the drivable area. Its unique architecture composed of unified transformer-based modules allows features extracted from multi-camera inputs to enable consistent spatial-temporal reasoning across 3D object detection and lane segmentation. This consistency ensures that detected objects and lane boundaries are spatially and temporally aligned, addressing some of the limitations of previous work in Section II-A. Still, 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 *Scene Graph (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., infront, behind, nextto) and distance-based relationships (e.g., 1m, 2m) are computed relative to the ego vehicle's position. This structured approach ensures that the scene graph captures all critical relationships necessary for downstream reasoning, providing a compact, interpretable, and semantically rich structure that enables efficient evaluation of driving properties and supports the correction mechanism.

C. 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 II-B [11], [5] to determine whether the precondition holds over the graph. Given the SG, sg, $\Phi_{active} = \{\phi \,|\, \phi \in \Phi, \phi_{\mathcal{X}}(sg)\}$.

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 3. From Section III-A, 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.

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 3; 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 3; 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.

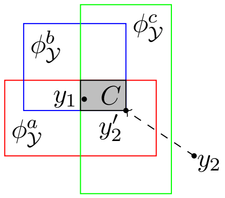

Fig. 3: Constraint Intersection, C, for $\phi^a_{\mathcal{Y}}$, $\phi^b_{\mathcal{Y}}$, $\phi^c_{\mathcal{Y}}$ shaded in gray and the correction applied to outputs y_1 and y_2.

IV. 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 is fundamentally tied to 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. Preliminary work exploring the tradeoff between monitoring frequency and corrective effectiveness indicates that even when operating at a 1 Hz rate, M4PC (SGG) demonstrates its potential by reducing safety-critical infractions compared to the base implementation [4] (Chapter 5.3).

Second, M4PC's definition of what constitutes desirable behavior is expressed exclusively by the properties provided. It will correct for those properties at the cost of any unstated properties. The definition of what constitutes an adequate set of properties from the perspective of correction is a topic for future work.

V. Evaluation

This study aims to assess M4PC's effectiveness in reducing driving property violations through runtime correction.

A. Study setup

Systems under Monitoring. We evaluate three autonomous driving systems. The first two, TCP [26] and Interfuser [27], have been widely adopted in the literature as strong baselines [28] and were also used as the units of analysis in prior work on T4PC [5]. 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 [29], 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 [30] and Apollo [31]. 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.

Properties. To evaluate the effectiveness of M4PC, we employ a set of six driving properties formalized in RFOL and summarized in Table I. 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 T4PC [5], 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_5–ϕ_6, as these rules were already identified in T4PC to effectively characterize the stop-sign related behaviors where TCP exhibits violations. Finally, for Pylot [29], we use the combination of ϕ_1, ϕ_2, and ϕ_4–ϕ_6, because the original system is prone to running red lights, missing stop signs, and colliding with other vehicles.

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. Town 05 features diverse road topologies, including two-lane urban streets, multi-lane highways, and T-intersections, as well as a wide range of dynamic entities such as traffic lights, stop signs, pedestrians, cyclists, cars, and trucks. 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.

Techniques. For two of the three systems under test (TCP and Interfuser), we consider the finetuned baselines released by T4PC [5] and the corresponding T4PC models trained with property loss. For Pylot, we evaluate only the original system, and we then apply our new approach, M4PC, to all three systems. While T4PC is the state-of-the-art for property-driven compliance, 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 scene graphs extracted directly from the simulator using a CARLA plugin [22], and another using M4PC's novel scene graph generator based on UniAD. 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). 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.

B. Results

Table III reports the results for TCP, Interfuser, and Pylot. Across the three systems, M4PC demonstrates clear benefits, with the ground-truth (GT) variant consistently achieving the highest driving scores and the lowest rate of critical infractions. Our raw data and extended tables are available at https://github.com/less-lab-uva/m4pc

For TCP, M4PC (GT) achieves an average driving score of 87.16, substantially outperforming both the TCP baseline (76.86)—deemed significant by an ANOVA with $p = 0.0504$ with Bonferroni correction—and TCP T4PC (78.23). This improvement stems primarily from a dramatic reduction in stop sign infractions, which decrease from 1.10 in the baseline and 0.94 in T4PC to only 0.12 on average under M4PC (GT). The SGG-based variant of M4PC also demonstrates strong performance with a driving score of 79.99 and commits only one-fifth the stop sign infractions of both TCP baseline and TCP T4PC. While M4PC (SGG) effectively enforces stop sign compliance through properties ϕ_5 and ϕ_6, 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

TABLE III: TCP, Interfuser, and Pylot results. Values in bold are the best.

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**
TCP 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
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**
Interfuser 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)	60.14±32.37	0.12±0.39	0.54±0.79	0.42±0.64	0.04±0.20	0.00±0.00	0.12±0.33
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**

additional collision-prevention properties, which we omitted here to ensure fair comparison with T4PC's property set.

For Interfuser, the trend is similar. M4PC (GT) achieves the highest driving score (69.74), improving over both the baseline (59.41) and T4PC (64.05). It reduces collisions with pedestrians over 61% and 55% compared to the baseline and T4PC models, respectively. Similarly substantial drops are observed for collisions and red light infractions.

The SGG variant trails behind GT with a score of 60.14. It reduces pedestrian and vehicle collisions as well compared to baseline and T4PC models. It is also worth noting that M4PC reduces the route timeouts compared to the T4PC-trained model, which would get stuck more often as a side effect of its retraining for property compliance. The reduced improvements relative to GT occur because UniAD struggles to detect very close objects, thus M4PC (SGG) cannot reliably avoid nearby vehicles approaching from the side. Additionally, vehicle blocked infractions increase because in some scenarios entities get stuck in front of the ego vehicle and M4PC prevents further motion, whereas the original model would slowly push forward, collide, and continue along the route. This increase is not a limitation, but rather a sign that M4PC prioritizes safety by halting the vehicle to avoid risky maneuvers or collisions. In contrast, baseline systems may continue and cause infractions. Addressing blocked scenarios is the responsibility of the planner, not the monitor; integrating re-planning could further reduce timeouts and improve route completion.

For Pylot, M4PC also delivers substantial 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 37%, red light infractions by 29%, and stop sign infractions by 100%. The SGG variant also improves over the baseline with a driving score of 75.38, reducing vehicle collisions by 25%, and stop sign infractions by 50%. However, it does not substantially reduce red light infractions despite having property ϕ_2, instead demonstrating similar performance to the baseline. 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.

Overall, the results highlight three key findings. First, runtime correction via M4PC is consistently effective across different system architectures, reducing safety-critical violations and improving overall driving performance. Second, while ground-truth scene graphs unsurprisingly yield the best outcomes, the SGG variant already provides competitive performance, 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 or superior performance, thereby establishing M4PC as a complementary technique that can enhance compliance without retraining models. Importantly, unlike T4PC, our method does not require access to model weights nor incurs model retraining costs, yet it achieves similar improvements through runtime correction alone. This analysis represents an important first step in applying M4PC to diverse systems, demonstrating its potential to improve property compliance; 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.

VI. Conclusion

We have introduced M4PC, a novel approach to detect and correct driving violations in autonomous vehicles. Our method operates independently of the system's internal workings, making it more generally applicable than existing techniques. Preliminary results suggest that our current implementation can improve a baseline system, while our most optimized implementation could outperform state-of-the-art techniques that require system access. Future work includes tackling properties that require correcting a sequence of actions over time and deploying the approach in a real-world setting to shadow an AV.

VII. Acknowledgments

This research is supported in part by NSF awards 2312487 and 2403060; by ARO grant W911NF-24-1-0089; and by Lockheed Martin Advanced Technology Labs. The authors acknowledge Research Computing at The University of Virginia for providing access to computational resources.

References

[1] A. Dinamani, May 2025, accessed on 09.10.2025. Link.

[2] A. Marshall, "Uber video shows the kind of crash self-driving cars are made to avoid," Mar 2018, accessed on 07.25.2025. Link.

[3] N. E. Boudette and N. Chokshi, "U.s. will investigate tesla's autopilot system over crashes with emergency vehicles," *New York Times*, Aug 2021, accessed on 07.25.2025. Link.

[4] F. Toledo, "SD4AS: Safe driving for autonomous systems," PhD dissertation, University of Virginia, School of Engineering and Applied Science, 2026.

[5] F. Toledo, T. Woodlief, S. Elbaum, and M. B. Dwyer, "T4pc: Training deep neural networks for property conformance," *IEEE Transactions on Software Engineering*, 2025.

[6] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018.

[7] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 132–142.

[8] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: automated whitebox testing of deep learning systems," *Commun. ACM*, vol. 62, no. 11, p. 137–145, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3361566

[9] Y. Sun, C. M. Poskitt, J. Sun, Y. Chen, and Z. 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.

[10] H. Qi, S. Chen, F. Zhang, T. Tsuchiyak, M. Hayashi, M. Okada, L. Ma, and J. Zhao, "Conflict-based scenario generation for autonomous driving system validation," in *2025 IEEE/ACM 1st International Workshop on Software Engineering for Autonomous Driving Systems (SE4ADS)*. IEEE, 2025, pp. 5–11.

[11] T. Woodlief, F. Toledo, S. Elbaum, and M. B. Dwyer, "The sgsm framework: Enabling the specification and monitor synthesis of safe driving properties through scene graphs," *Science of Computer Programming*, vol. 242, p. 103252, 2025.

[12] T. Woodlief, F. Toledo, M. Dwyer, and S. Elbaum, "Scene flow specifications: Encoding and monitoring rich temporal safety properties of autonomous systems," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 2524–2547, 2025.

[13] Y. Sun, C. M. Poskitt, X. Zhang, and J. Sun, "Redriver: Runtime enforcement for autonomous vehicles," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024.

[14] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann, "A comprehensive survey of scene graphs: Generation and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[15] H. Li, G. Zhu, L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, X. Zhao, S. A. A. Shah, and M. Bennamoun, "Scene graph generation: A comprehensive survey," *Neurocomput.*, vol. 566, no. C, mar 2024.

[16] C. Morse, L. Feng, M. Dwyer, and S. 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.

[17] A. V. Malawade, S.-Y. Yu, B. Hsu, H. Kaeley, A. Karra, and M. A. Al Faruque, "Roadscene2vec: A tool for extracting and embedding road scene-graphs," *Know.-Based Syst.*, vol. 242, no. C, apr 2022.

[18] F. Toledo, S. Elbaum, D. Gopinath, R. Kaur, R. Mangal, C. S. Păsăreanu, A. Roy, and S. Jha, "Monitoring safety properties for autonomous driving systems with vision-language models," in *2025 IEEE Engineering Reliable Autonomous Systems (ERAS)*, 2025, pp. 1–8.

[19] J. Yang, Y. Z. Ang, Z. Guo, K. Zhou, W. Zhang, and Z. Liu, "Panoptic scene graph generation," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVII*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 178–196. [Online]. Available: https://doi.org/10.1007/978-3-031-19812-0_11

[20] Y. Cong, M. Y. Yang, and B. Rosenhahn, "Reltr: Relation transformer for scene graph generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 11 169–11 183, 2023.

[21] R. Li, S. Zhang, D. Lin, K. Chen, and X. 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. 28 076–28 086.

[22] T. Woodlief, F. Toledo, S. Elbaum, and M. B. Dwyer, "Closing the gap between sensor inputs and driving properties: A scene graph generator for carla," in *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2025, pp. 29–32.

[23] ——, "S3c: Spatial semantic scene coverage for autonomous vehicles," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639178

[24] V. D. of Motor Vehicles, "Virginia driver's manual," accessed on 04.23.2025. Link.

[25] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, L. Lu, X. Jia, Q. Liu, J. Dai, Y. Qiao, and H. Li, "Planning-oriented autonomous driving," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 17 853–17 862.

[26] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. 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*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2024.

[27] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," in *Conference on Robot Learning*. PMLR, 2023, pp. 726–737.

[28] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10 164–10 183, 2024.

[29] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. 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.

[30] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. 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.

[31] "Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2025-09-11."