SavviDriver: Model-based Framework for Game-based Testing of Autonomous Vehicles in Diverse Multi-agent Traffic Scenarios

Kenneth H. Chan[†], Sol Zilberman[†] and Betty H.C. Cheng[†]

Department of Computer Science and Engineering, Michigan State University, 428 S Shaw Ln, East Lansing, 48824, MI, USA.

*Corresponding author(s). E-mail(s): chanken1@msu.edu;
Contributing authors: zilberm4@msu.edu; chengb@msu.com;
[†]These authors contributed equally to this work.

Abstract

Autonomous vehicles (AVs) must operate safely in the face of uncertainty, including those induced by human behaviors (i.e., external human drivers). Specifically, AVs must exhibit safe responses when encountering previously unseen behaviors from human drivers with different driving styles. For example, aggressive drivers may cut off other vehicles to merge into a lane, or distracted drivers may fail to respond to changing road conditions. A key challenge is how to assess and improve the onboard AV decision-making capabilities to detect and mitigate those potentially unsafe scenarios due to one or more external human-operated vehicles. We observe that the AV and the other vehicles on the roadway may share common functional objectives (e.g., to navigate to a given target destination), but otherwise may be motivated by different non-functional objectives, such as safety, minimizing transport time, minimizing fuel consumption, etc. This paper introduces a modular and composable model- and gaming-based testing framework to enable an AV developer to operationally assess and even improve the robustness of an AV in response to human-based uncertainty. Specifically, this work uses goal models to declaratively specify functional and non-functional objectives of vehicles (both the AV under study and those representing external human-operated vehicles) to inform the Reinforcement Learning (RL) realization of the gaming environment that incorporates real-world traffic infrastructure data. We demonstrate the model-based capabilities of our game-based testing approach on a number of scenarios based on real-world traffic accident data involving human drivers.

1 Introduction

Increasingly, Autonomous Vehicles (AVs) are being deployed alongside human-driven vehicles (i.e., mixed traffic environments [1]), where they must safely interact with other road users with different behaviors, objectives, and driving styles. To ensure operational correctness and prevent failures, developers must understand how AVs respond to different sources of uncertainty, including the potentially unpredictable behaviors of human drivers. However, significant safety concerns and high resource costs limit "in the field" testing of safety-critical systems in real-world settings [2]. Thus, there is growing interest in the use of human-like driver models and simulations to test AVs in representative operating contexts, including elements that pose uncertainty, such as mixed traffic interactions [3]. This paper proposes a modular, goal model-based framework that harnesses the non-cooperative gaming paradigm [4] to explore the behavior of interacting agent vehicles, including AVs and human-operated vehicles, in order to assess and improve the robustness of AVs in response to human-based uncertainty.

Uncertainty arises when a system encounters an event it cannot handle, often due to incomplete information (i.e., epistemic doubt [5]) and/or unpredictable phenomena in its operating environment (i.e., aleatoric uncertainty [6, 7]). *Human-based uncertainty* for AVs in mixed-traffic environments results from epistemic doubt about other drivers' intentions, such as motivations, behaviors, and future decisions, during design-time testing and run-time decision-making. Recently, several state-of-the-art *game-based testing* approaches [8, 9, 10, 11, 12] have used game-theoretic formulations of traffic scenarios to assess AV responses to uncertainty posed by interactions with other vehicles; Reinforcement Learning (RL) [13] is often used to operationally realize various types of vehicle behavior [8]. However, game-based testing techniques commonly rely on ad-hoc approaches (e.g., trial-and-error) to determine game objectives, such as RL-based reward functions, and are often tightly coupled to specific traffic scenarios. Both of which require low-level manual code changes and/or brute-force approaches to explore any operational context changes, including variations on the number and types of drivers, road scene changes, etc. [9]. As such, existing state-of-the-art game-based testing approaches lack systematic or automated means to assess AV robustness in the face of human-based uncertainty.

This work introduces SavviDriver (**S**afe **A**utonomous **V**ehicle-to-Human-Controlled **V**ehicle **I**nteractions), a modular and composable goal model-based framework for game-based testing of AVs in diverse multi-agent traffic scenarios. Several key insights are foundational to our approach. First, goal models can be used to declaratively specify and manage game-based testing of multi-agent interactions with respect to functional and non-functional objectives. Second, by refining goal models down to individual requirements that can be assessed for satisfaction/satisficement in terms of utility functions [14], we can use the utility functions to specify the reward structure for RL-based realization of a game-based testing framework. Finally, by taking a "separation of concerns" approach [15, 16] to goal modeling, we can separate the context-dependent functional goals from context-independent non-functional goals. This strategy allows developers to *decompose* top-level goals into a library of reusable components (i.e., subtrees) corresponding to different driving styles (e.g., aggressive driving non-functional subtree). Then a developer can *compose* different combinations and structures of subtrees from the library to facilitate the rapid reconfiguration of multi-agent game-based testing with heterogeneous/homogeneous agents.

SavviDriver supports the assessment and improvement of AV robustness in response to uncertainty posed by one or more human-based agents through game-based testing. Consider a developer who is interested in exploring the interaction(s) between the AV under development and different types of human driver models to discover unexpected outcomes. First, a developer defines the participating game actors (road users such as AV, human-driven vehicles, etc.) and the environmental context (e.g., merging traffic scenario) in which they interact. A developer may declaratively specify different types of human-based driving styles (e.g., aggressive, distracted, timid, etc.) based on real-world traffic data [17, 18, 19], as well as intended AV behavior(s) (e.g., safety, comfort, etc.). Next, SavviDriver uses a variation of the KAOS goal modeling language [20] to explicitly define the functional and non-functional objectives of the actors in the game. SavviDriver supports the reuse of behavior-based templates for goal model subtrees corresponding to different human-based driving styles (e.g., an 'aggressive' driving subtree can be used in different mixed-traffic scenarios). We associate utility functions [21] with leaf-level goals to assess their satisfaction [22]. The utility functions can be used to inform the objective function for the corresponding agent in multi-agent games. As the agents for the AV and human-operated vehicles pursue their respective goals while operating in the traffic scenarios under study (e.g., lane merging), unexpected and unsafe behavior may be exhibited by one or more of the agents due to their unanticipated interaction(s). The discovered behavior of both AV and human-operated vehicles can be used to assess and potentially improve the robustness of the AV and inform changes to goal models to explore additional mixed-traffic interactions, rather than editing low-level simulation code.

To demonstrate how SavviDriver can be used to (1) discover unexpected interactions between AVs and external agents in the environment and (2) improve the robustness of the AV with respect to human-based uncertainty, we have applied it to several use cases that capture real-world traffic accident data [18, 23]. The remainder of this paper is organized as follows. Section 2 provides background information and overviews enabling technologies. Section 3 describes the SavviDriver framework. Section 4 shows the results of our illustrative use cases. Section 5 discusses our results, and Section 6 provides the threats to validity. Finally, Section 7 concludes this paper and discusses future work.

2 Background

This section describes background topics and enabling technologies used in SavviDriver. First, we overview existing assurance challenges for AVs. Next, we discuss existing state-of-the-art approaches involving game theory and RL to discover uncertainty for AVs. Finally, we describe the goal modeling language and utility functions used in this work.

2.1 Challenges for Autonomous Vehicles

Advances in machine learning have improved the capabilities of AVs. Recently, a number of different companies have deployed a range of AVs in real-world settings, including Tesla's Autopilot [24], Waymo's Taxi services [25], and ADASTEC's autonomous bus services [26]. To avoid accidents and protect their occupants, deployed AVs must demonstrate safe behavior in addition to satisfying operational constraints. Based on Waymo's 2023 safety report, AVs have the potential to reduce the frequency and severity of traffic accidents in limited operating contexts [27]. However, various uncertainty factors, including those introduced by machine learning and humans, have been shown to cause unexpected behaviors in AVs [28]. For example, human drivers exhibit a range of driving styles, including aggressive, distracted, or even malicious behaviors (e.g., brake-checking, tailgating, etc.) [29]. Vehicles with these driving styles have been associated with unsafe driving maneuvers on the road, such as aggressively cutting off another vehicle or drifting out of a lane due to distractions [30, 31]. In order to ensure AVs are capable of reacting safely when encountering these unexpected maneuvers, AVs must be exposed to a wide range of human-based behaviors during training and testing [28].

2.2 Game Theory for Uncertainty Exploration

In order to ensure safe behaviors, a developer may explore a number of techniques to test the AV before deployment, including human-based testing in a simulation environment [32, 33, 34]. However, human-based testing faces the challenges of testing bias and incurs expensive development time and effort [35]. One promising approach to model human behavior is through game theory that enables the modeling and analysis of strategic interactions between *rational decision-makers* [36]. In game theory, a game involves a number of agents that interact with each other in an environment. Agents seek to achieve individual and/or shared objective(s). In *cooperative* game theory, agents can collaborate to maximize collective rewards [36]. However, as AVs are deployed in real-world traffic and interact with traditional human-operated vehicles, it is not feasible to assume collaboration. *Non-cooperative* game theory [4] better captures the relationship between road users than cooperative game theory, as drivers are mainly concerned with their individual objectives and do not (typically) maliciously interact with other road users [8]. Recently, several state-of-the-art gaming approaches have used *RL* to operationalize non-cooperative games between vehicles in traffic simulations to test AVs before deployment [8, 9, 11]. Chan *et al.* [8] proposed the combination of non-cooperative game theory and RL to discover previously unseen or undesirable behaviors for AVs in two-player games. Wachi *et al.* [11] and Hao *et al.* [9] demonstrate different ways that multi-agent games and adversarial RL [37] can be used to discover failure cases for rule-based vehicles in simulation. However, existing game-based testing techniques largely use ad hoc and/or hard-coded techniques to generate traffic scenarios, driving styles, and RL agents, where any changes (e.g., different roadways, the number/types of drivers, or reward structures) may require significant development effort.

2.3 Reinforcement Learning

RL is a learning-based approach, where agents learn to perform actions in an environment that maximizes a reward function. Figure 1 shows a high-level overview of RL, where an agent performs some action affecting the environment. The updated state of the observable environment and the reward associated with the state are then returned to the agent. The reward function informs the behavior(s) of the agent, motivating it to achieve a given task. For example, an agent whose objective is to complete a racing game may have a reward function that motivates it to complete the race as fast as possible, stay on the track, and avoid collisions [38]. Specifically, an agent in RL learns a *policy* that represents a mapping between states of the environment and *optimal* actions in each of those states. Learning a policy involves a "trial-and-error" process, where agents make iterative improvements to the current policy by interacting with an environment over a given number of trials. During each trial (game), an agent may discover one or more actions that may improve the overall reward. An optimization algorithm informs the updates to the policy after each trial. Recent advances in Deep Reinforcement Learning (DRL) have demonstrated that Deep Neural Networks (DNNs) can be used to approximate the optimal policy for a given agent [39]. DRL is often used for complex tasks such as driving that may require agents to optimize high-dimensional non-linear objective functions in dynamic environments [40].

In the context of RL-based behavior generation, a challenge is how to design a reward function that accurately rewards and motivates a desired behavior. Existing approaches often use ad-hoc development approaches to design those rewards, resulting in reward functions that are often difficult to manage or interpret and may be overly complex [41]. In addition, traditional approaches require "trial-and-error" low-level code changes to explore how different reward configurations result in different behaviors. In contrast, SavviDriver provides a systematic means to structure the reward function with support for updates via high-level model changes.

2.4 Goal-Oriented Requirements Engineering

Van Lamsweerde *et al.* introduced KAOS, a goal-oriented approach to requirements engineering, where a high-level goal or objective is decomposed into subgoals. Each subgoal is then recursively decomposed until a low-level leaf goal is reached [20]. At the leaf level, KAOS goals are considered requirements that are discharged to *system components* for satisfaction or satisficement (i.e., degree of satisfaction) [20].[1] While KAOS does not distinguish functional and non-functional goals explicitly, subgoal decomposition strategies can be used to specify functional goals that reflect non-functional properties. Effectively, our KAOS goal model captures the non-functional properties of driving styles in terms of the functional goals/subgoals. For example, the non-functional subgoal, '`drive safely`', can be refined down to requirements: '`driving at the speed limit`' and '`maintaining a minimal trailing distance`'. Those requirements can then be handled by the system components '`speed controller`' and '`radar`', respectively.

[1] We use the term "system components" instead of "agents" in the KAOS goal model to avoid confusion with the use of the term "agent" for AV and non-ego vehicles.

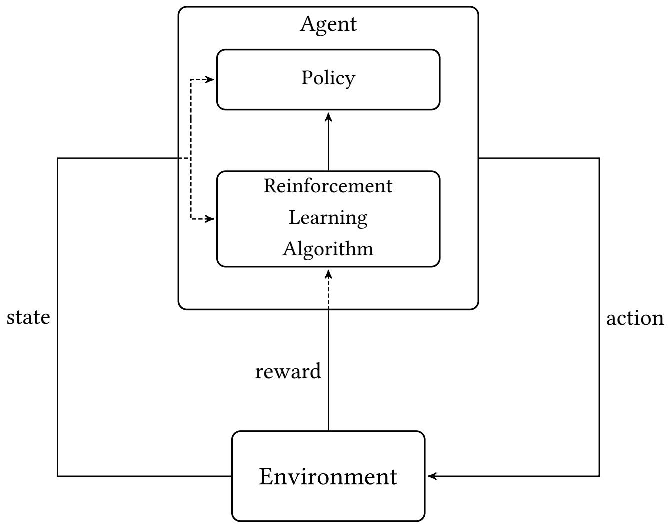


Fig. 1: A high-level overview of RL [42, 43].

2.5 Utility Functions

Utility functions map system and environmental attributes to quantitative values that establish a degree of system goal satisfaction [22, 44, 45, 46]. Expression (1) shows the general structure of a utility function, where u represents a scalar utility value between $[0, 1]$, and v represents a measured property obtained from the system and its environment (e.g., vehicle speed, distance to target destination) [22].

$$u = f(v) \tag{1}$$

Previously, it has been shown that *utility functions* can be used to annotate KAOS leaf-level goals to measure their satisficement [22, 45, 47]. Specifically, leaf-level goals are annotated with utility functions that specify how the corresponding system component can be used to measure a quantifiable attribute of the leaf-level goal. For example, the requirement '`driving at the speed limit`' can be quantified by the utility function shown in Expression (2), where s corresponds to the speed limit and v corresponds to the agent's current velocity. The maximum value (i.e., *utility*) occurs when the agent is driving exactly at the speed limit (i.e., $s - x = 0$).

$$f(x) = \begin{cases} 0 & \text{if } v < .95s, v > 1.05s \\ |\frac{s-v}{0.05s}| & \text{if } .95s \leq v \leq 1.05s \end{cases} \tag{2}$$

3 Methodology

This section describes the SavviDriver framework, including details of its modeling and analysis processes. Table 1 provides a high-level glossary of the terms and their definitions used in this work. Figure 2 shows an overview of the SavviDriver framework, where circles depict processes, parallel lines depict persistent data stores, labeled arrows depict data flow between entities, and rectangles depict external entities. We next describe the elements of SavviDriver in detail.

Table 1: Overview of the terminologies used in this paper.

Term	**Definition**
ITS	An Intelligent Transportation System (ITS) comprising one or more intelligent vehicle(s) (e.g., AVs) as well as one or more human-operated vehicle(s).
Agents	Entity with a concrete role (e.g., vehicle) that has goals and is of interest in the ITS under study.[†]
AV	An autonomous vehicle whose behavior is optimized by one or more machine learning component(s).
Ego	The autonomous vehicle under study.
NeV	Human-operated vehicle that is part of the operational context for the AV in the ITS.
Functional goal	Goals that describe functions that the system should perform (e.g., arrive at destination).
Non-functional goal	Goals that describe desired properties (e.g., defensive driving) of *how* a system satisfies its functional goals.
Scenario Definition	Concrete parameters of the identified traffic scenario, including infrastructure data and initial agent states.
Scenario Objective	The individual functional objectives of each agent in a given scenario.
Driver Type	The driving style (e.g., aggressive, cautious, etc.) and observed behaviors (e.g., speeding, swerving, etc.) of each agent.

[†] The term *agent* is common to the disparate techniques used in this work (e.g., goal modeling, reinforcement learning, game theory). We use the definition presented in this table whenever the term *agent* is used.

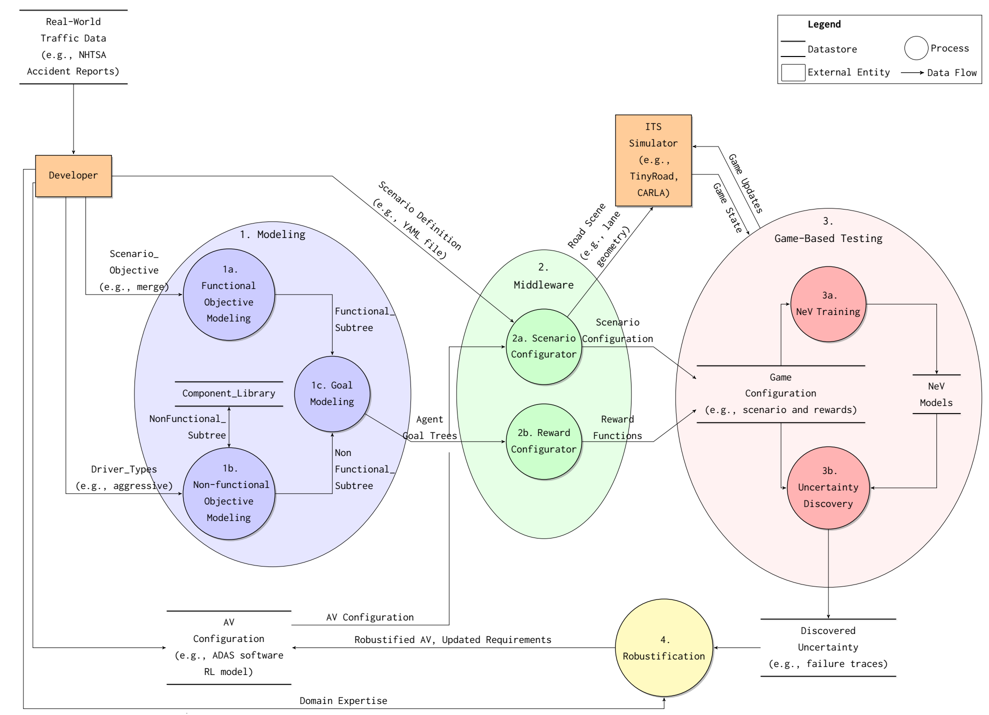

Real-World Traffic Data (e.g., NHTSA Accident Reports)
Legend
Datastore
Process
External Entity
Data Flow
Developer
ITS Simulator (e.g., TinyRoad, CARLA)
Scenario Definition (e.g., YAML file)
Road Scene (e.g., lane geometry)
Game Updates
Game State
1. Modeling
Scenario_ Objective (e.g., merge)
1a. Functional Objective Modeling
Functional_ Subtree
1c. Goal Modeling
Component_Library
NonFunctional_ Subtree
Driver_Types (e.g., aggressive)
1b. Non-functional Objective Modeling
Non Functional_ Subtree
Agent Goal Trees
2. Middleware
2a. Scenario Configurator
2b. Reward Configurator
Scenario Configuration
Reward Functions
3. Game-Based Testing
3a. NeV Training
Game Configuration (e.g., scenario and rewards)
NeV Models
3b. Uncertainty Discovery
AV Configuration (e.g., ADAS software RL model)
AV Configuration
Robustified AV, Updated Requirements
4. Robustification
Discovered Uncertainty (e.g., failure traces)
Domain Expertise

Fig. 2: Overview of SavviDriver. Blue processes represent manual modeling steps done by a `Developer`. Green processes represent an automated middleware compilation of `Developer` information. Red processes represent game-based testing components. Finally, the yellow process represents the AV robustification step of the framework.

3.1 Data

Historical data, documentation, and domain expert input regarding driving data and traffic accidents are all used by the developer to identify and specify different types of driver intentions. As such, SavviDriver can be used to explore human-based uncertainty found in real-world traffic scenarios [17, 18, 19]. Specifically, during pre-processing, domain experts identify through manual or automated techniques (e.g., accident analysis procedures [19]) three key types of information:

- **Scenario Definition**: Concrete parameters of the identified traffic scenario that describe the structural attributes (e.g., lane geometry, topography, etc.), regulatory attributes (e.g., speed limit, direction of the lane) of the traffic infrastructure, and the participating vehicles (e.g., type, initial position, orientation, and velocity of each vehicle).
- **Scenario Objective:** The functional objective of each agent participating in a given scenario, as identified by domain experts. For a given scenario, each agent's

scenario objective may capture scenario-specific maneuvers an agent should perform (e.g., merge into right lane), with respect to their initial state in the traffic infrastructure.
- **Driver Types**: The driving style (e.g., aggressive, cautious, etc.) and observed behaviors (e.g., speeding, swerving, etc.) of each agent.

Running Example: We illustrate the SavviDriver framework using a running example of a merge scenario, motivated by United States National Highway Traffic Safety Administration (NHTSA)'s 2022 accident reports [17, 48] that list lane merging as a frequent cause of accidents. Figure 3 provides a 2D illustration of the Intelligent Transportation System (ITS) under study. The scenario under study is a two-lane road that converges into a single lane. The participating agents include the blue Ego (the AV under study) driving in the right lane and an orange aggressive Non-ego Vehicle (NeV) that must merge into the Ego's lane.

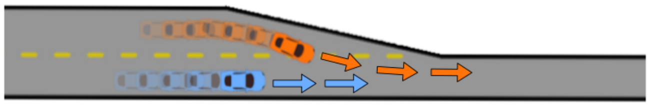


Fig. 3: Merge scenario sample showing the Ego (blue) and NeV (orange) in the TinyRoads simulator [8].

3.2 Step 1. Modeling

This step describes how agents are modeled by the `Developer` in SavviDriver; this process is used to individually develop respective goal models to describe Ego and NeV(s) behaviors. Modeling Process 1 overviews the `Developer`'s process for goal modeling promoted by SavviDriver. We use a variation of the KAOS goal modeling language to elaborate the top-level functional objective of each agent (e.g., '`arrive at destination`'). A `Developer` uses KAOS to create an AND/OR goal model to declaratively specify at a high-level *what* an agent's objectives are and the driving style (i.e., non-functional requirements) they should exhibit when satisfying those objectives. We next describe the elements of the goal model-based decomposition strategy, which is used to model both the behavior of the Ego and NeV(s).

For each agent, the `Developer` refines and decomposes the agent's top-level objective into two distinct types of subtrees: functional subtrees and non-functional subtrees. Figure 4 shows an abstract overview of goal decomposition in SavviDriver, where the top-level goal is shown in yellow, *KAOS Goals* are depicted by parallelograms, and goal decomposition is represented by refinement arrows. The blue subtree in Figure 4 shows **Step 1a** of SavviDriver, where the `Developer` defines the *context-dependent functional objectives* that correspond to a `Scenario_Objective`. For example, a vehicle may have a `Scenario_Objective` of merging into the right lane of the roadway if its initial position is in a merging lane (see orange vehicle, Figure 3). The red subtree in Figure 4 shows **Step 1b** of SavviDriver, where the `Developer` defines the *context-independent* *non-functional objectives* corresponding to a `Driver_Type` that captures the driving style(s) a vehicle may exhibit while completing its functional objectives. For example, a vehicle may exhibit aggressive or distracted driving styles while attempting to merge into the right lane of the roadway. Finally, in **Step 1c**, the `Developer` composes the two subtrees from **Step 1a** and **Step 1b** to form the overall KAOS goal tree for the agent.

Modeling Process 1: Step 1 of SavviDriver (performed by developer).

```
1:  /* Step 1a. Functional Objective Modeling */
2:  ▷ Decompose Scenario_Objective down to requirements to define Functional_Subtree.
3:  ▷ Annotate leaf-level requirements with utility functions.
4:
5:  /* Step 1b. Non-functional Objective Modeling */
6:  ▷ If Driver_Type exists in Component_Library.
7:        ▷ Retrieve NonFunctional_Subtree from existing Component_Library.
8:  ▷ Else
9:        ▷ Decompose Driver_Type down to requirements to define NonFunctional_Subtree.
10:       ▷ Annotate leaf-level requirements with utility functions.
11:       ▷ Add NonFunctional_Subtree to Component_Library.
12:
13: /* Step 1c. Goal Modeling */
14: ▷ AND-compose Functional_Subtree and NonFunctional_Subtree and attach to top-level goal.
15: ▷ Return agent goal model.
```

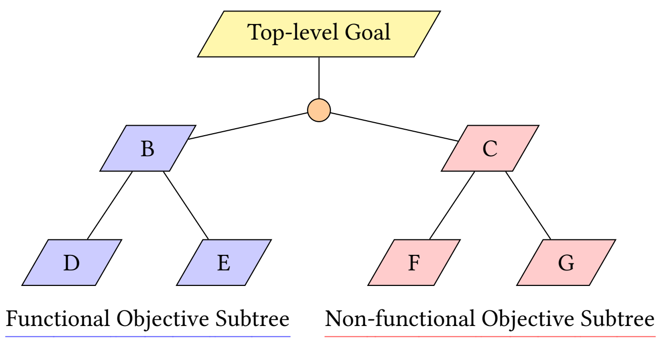

Top-level Goal
B
C
D
E
F
G
Functional Objective Subtree
Non-functional Objective Subtree

Fig. 4: Abstract representation of the KAOS goal tree for a vehicle in a scenario. The functional subtree decomposition is shown in blue, while the non-functional subtree decomposition is shown in red.

SavviDriver's goal modeling approach facilitates and promotes modularity and reusability of agent objectives. Specifically, high-level agent objects can be decomposed into a library of reusable *components* corresponding to different driving styles, represented by context-independent subtrees. Figure 5 shows how those components can be reused and composed to explore different mixed-traffic scenarios. The first row (I.) shows the merge scenario with subtree B-D-E with two different non-functional subtree decompositions: aggressive and distracted. The second row (II.) shows the extraction of the non-functional subtree of each merge scenario and their addition to the `Component Library`. The third row (III.) shows that the subtrees from the `Component Library` can be reused in a different scenario (i.e., left-turn). As such, SavviDriver supports the interchange and reuse of subtrees and scenarios to discover uncertainty (e.g., unexpected interactions between an Ego and different combinations of NeVs in diverse mixed-traffic scenarios).

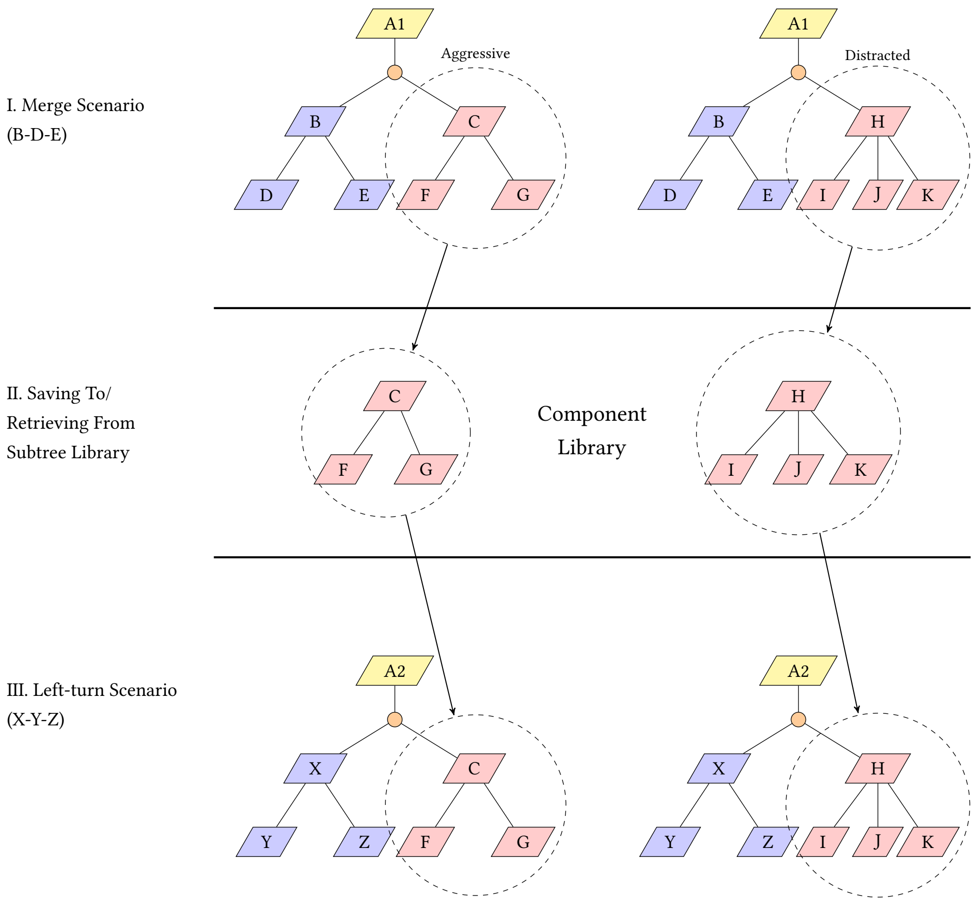


Fig. 5: Example of how modular subtrees can be reused in SavviDriver. The first row (I.) shows the merge scenario with two different non-functional objectives (i.e., aggressive (C-F-G) and distracted (H-I-J-K)). The second row (II.) shows the extraction of the subtrees from the merge scenario to the `Component Library`. Finally, the third row (III.) shows the left-turn scenario, reusing the non-functional objectives from the merge scenario with a new functional objective.

Consider the running example in Figure 3 involving the merge lane. We create goal models comprising their respective objectives for both the Ego and the NeV following the process described in **Step 1**. Figure 6a and Figure 6b show sample KAOS goal models for the Ego and Aggressive-NeV, respectively. *KAOS Goals* are depicted by parallelograms. Goal decomposition is represented by refinement arrows. Utility functions associated with leaf-level goals are represented by yellow ellipses, which are also used to specify objective functions for the driver model for each agent [22]. Green parallelograms denote goals annotated by fuzzy logic-based utility functions (i.e., satisficement) [45, 49], while blue parallelograms have boolean goals (i.e., either satisfied or not). The utility functions are discharged to system components, represented by white hexagons. We next describe the development of models for each agent in our running example in turn.

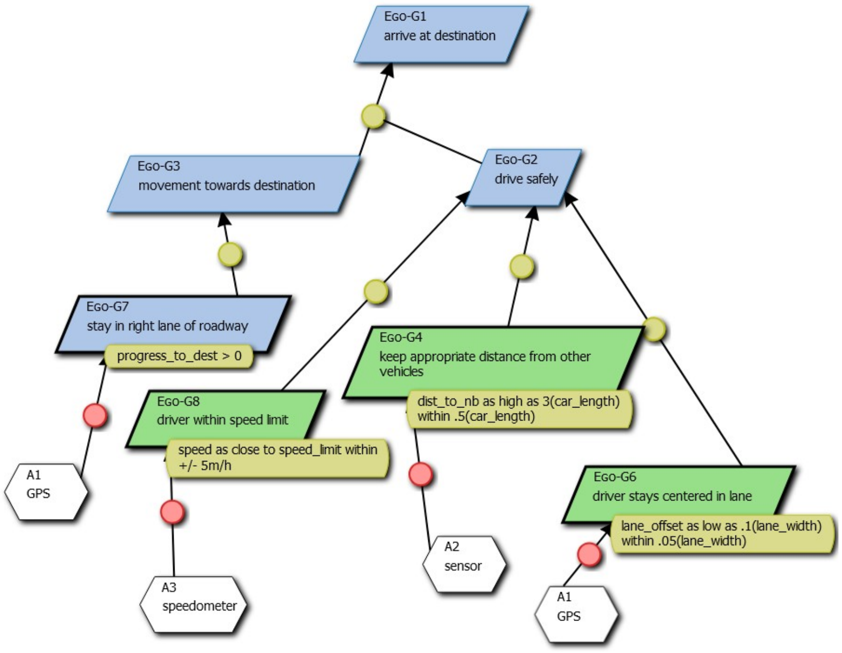
KAOS Goal Model for the AV (Figure 6a)

For the Ego, the top level goal Ego-G1 '`arrive at destination`' is `AND-` decomposed into the context-dependent functional goal Ego-G2 '`move towards destination`' (**Step 1a**) and context-independent non-functional goal Ego-G3 '`drive safely`' (**Step 1b**). The functional goal Ego-G2 is decomposed into ITS-specific goal Ego-G7 '`stay in right lane of roadway`'. The safety goal Ego-G3 is further decomposed into subgoals. Associated with each leaf-level goal (i.e., requirement) is a utility function that quantifies its satisficement (i.e., degree of satisfaction [20]). For example, the functional goal Ego-G4 '`keep appropriate distance from other vehicles`' is formally captured by the utility function in Expression (3), where x denotes the distance from the nearest vehicle and y denotes the vehicle's length. Finally, **Step 1c** composes subtrees rooted at goal Ego-G2 and goal Ego-G3 via KAOS `AND-` relation and adds them as children to the top-level goal Ego-G1.

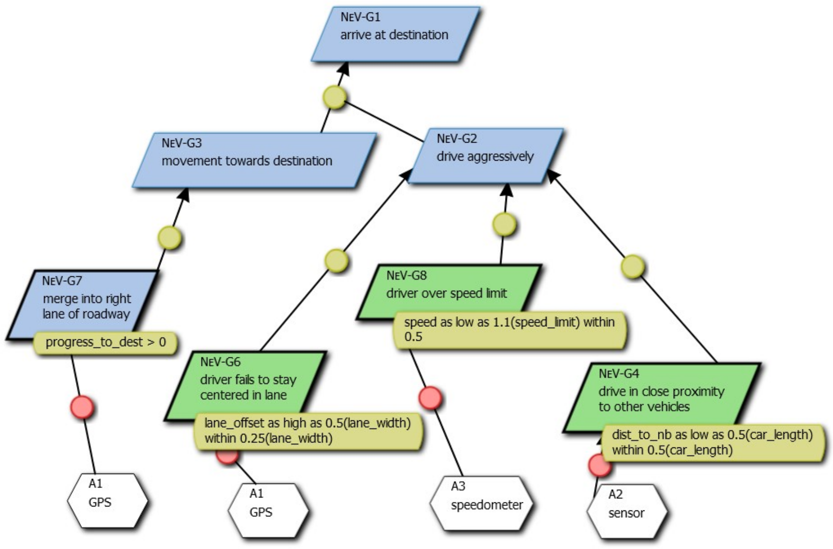
$$f(x) = \begin{cases} 0 & \text{if } x \leq 3.5y \\ 1 & \text{if } x >= 3y \\ \frac{3.5y - x}{0.5} & \text{if } 3y \leq x \leq 3.5y \end{cases} \tag{3}$$

KAOS Goal Model for Aggressive NeV (Figure 6b)

For the Aggressive-NeV, the top level goal NeV-G1 '`arrive at destination`' is `AND`-decomposed into the context-dependent functional goal NeV-G4 '`move towards destination`' (**Step 1a**) and context-independent non-functional goal NeV-G2 '`get to destination quickly`' (**Step 1b**). The functional goal NeV-G4 is next decomposed into ITS-specific goal NeV-G7 '`merge into right lane of roadway`'. The agent-specific goal NeV-G2 is then refined into a lower-level agent-specific goal NeV-G3 '`drive aggressively`'. The aggressive driving goal NeV-G3 is further `OR`-decomposed into quantified subgoals including speeding, swerving, and distance to vehicles. Finally, **Step 1c** composes subtrees rooted at goal NeV-G4 and goal NeV-G2 via KAOS `AND-` relation and adds them as children to the top-level goal NeV-G1 '`arrive at destination`'.


Ego-G1
arrive at destination
Ego-G3
movement towards destination
Ego-G2
drive safely
Ego-G7
stay in right lane of roadway
progress_to_dest > 0
Ego-G4
keep appropriate distance from other vehicles
dist_to_nb as high as 3(car_length) within .5(car_length)
Ego-G8
driver within speed limit
speed as close to speed_limit within +/- 5m/h
Ego-G6
driver stays centered in lane
lane_offset as low as .1(lane_width) within .05(lane_width)
A1
GPS
A3
speedometer
A2
sensor
A1
GPS

(a) KAOS goal model for the Ego.


NeV-G1
arrive at destination
NeV-G3
movement towards destination
NeV-G2
drive aggressively
NeV-G7
merge into right lane of roadway
progress_to_dest > 0
NeV-G8
driver over speed limit
speed as low as 1.1(speed_limit) within 0.5
NeV-G6
driver fails to stay centered in lane
lane_offset as high as 0.5(lane_width) within 0.25(lane_width)
NeV-G4
drive in close proximity to other vehicles
dist_to_nb as low as 0.5(car_length) within 0.5(car_length)
A1
GPS
A1
GPS
A3
speedometer
A2
sensor

(b) KAOS goal model for Aggressive-NeV.

Fig. 6: Overview of KAOS goal models for the Ego and the Aggressive-NeV.

3.3 Step 2. Middleware

This section overviews how SavviDriver's `Middleware` (**Step 2**) collates and processes information from the agent goal models and scenario specifications to coordinate the game-based testing of AVs. Specifically, the `Middleware` uses two different configurators, a `Scenario Configurator` (**Step 2a**) and a `Reward Configurator` (**Step 2b**), to automatically realize mixed-traffic interactions as a *game* between an Ego and one or more NeV(s), where driver behaviors are informed by agent-specific goal models. We next describe each configurator in turn.

2a. Scenario Configurator

Game theory provides the structure of the mixed-traffic scenario to support requirements-based exploration of human-based uncertainty. In our framework, a traffic scenario is captured as a game that comprises the following components: (1) the vehicles (i.e., players) of the game, (2) the actions available to each vehicle (e.g., steering, throttle), (3) the information structure (i.e., what a driver knows at a given step of the game), and (4) the objective function each driver seeks to maximize.

To this end, we formally define the mixed-traffic scenario as a non-zero-sum, non-cooperative game [4]. Let player set $\mathcal{I}$ represent N interacting players, where each player $p_i, p_i \in \mathcal{I}$ corresponds to a vehicle in the game (see Expression (4)).

$$\mathcal{I} = \{p_1, p_2, \ldots, p_N\}. \tag{4}$$

The gaming setup captures the number and type of agents, including the Ego and one or more NeVs, as well as the operating context in which they interact, such as a highway merge scenario or left-turn scenario. An `ITS Simulator` provides an environment that enables games to be executed to discover uncertainty. The `Scenario Configurator` (**Step 2a**) takes as input a `Scenario Definition` file to initialize the `ITS Simulator` and provide the `Scenario Configuration` as the operating context for the `Game Configuration`. A sample `Scenario Definition` file is shown in Listing 1. This information captures the road layout, the number and types of agents in play, and their initial states (e.g., position, velocity).

Listing 1: Sample `Scenario Definition` file for the merge scenario.

```yaml
map:
  filename: merge_road
vehicles:
  - name: Safety-Ego
    position: [427, 750]
    velocity: 40
  - name: Aggressive-NeV
    position: [370, 750]
    velocity: 40
```

2b. Reward Configurator

We next describe the automatic generation of each player's reward. The KAOS model serves as the "middleware" between the gaming setup and the ITS simulation, enabling developers to formally realize high-level developer input in terms of low-level functional goals instead of directly editing simulation code. In SavviDriver, non-cooperative game theory is used to support simulation-based testing to discover previously unseen traffic interactions between the Ego and one or more NeV(s). RL is used to realize the non-cooperative games, where gaming interactions are captured via RL reward functions. However, designing reward functions is challenging, especially for complex tasks with multiple competing objectives such as mixed-traffic scenarios [50]. Moreover, RL-based techniques often rely on ad-hoc development approaches that result in fragile/unstable reward functions; it may be unclear how modifying the reward function impacts observable agent behavior [41]. To this end, the `Reward Configurator` (**Step 2b**) "compiles" high-level agent objectives defined in goal models into RL reward functions, thereby bridging the gap between the previous modeling step and the non-cooperative ITS game, to discover uncertain behaviors for a given traffic scenario.

Recall that a scenario comprises a set of players p_i with corresponding KAOS goal models k_i. A strategy space $\mathcal{S}_i$ represents potential strategies s_i that player i can use for decision-making [51, 52]. In RL, a strategy s_i may be represented by a DNN that maps the current state of the operating environment observed by player p_i (e.g., vehicle locations, velocities, trajectories, etc.) to an action (e.g., actuator inputs such as braking, steering, etc.). Each player p_i seeks to find the "best" strategy s_i^* based on an individual reward (payoff) function π_i. Each player's DNN can be trained via RL to approximate the best strategy s_i^* [8, 13], which is formally specified in Expression (5):

$$s_i^* = \underset{s_i \in \mathcal{S}_i}{\operatorname{argmax}}\, \pi_i(s_i, s_{-i}^*), \forall s_i \in \mathcal{S}_i, \forall i \in \{1, \ldots, N\}, \tag{5}$$

where s_{-i}^* denotes the best strategies of all other players $\{p_1, ...p_{i-1}, p_{i+1}, ...p_N\}$. Thus, $\pi_i(s_i, s_{-i}^*)$ denotes the reward obtained by player p_i when using strategy s_i in the context of other players using strategies s_{-i}^*, respectively [52, 53].

For each vehicle agent (i.e., player p_i) in the scenario, SavviDriver automatically extracts utility functions from the associated KAOS goal, k_i, to structure its respective reward function π_i (see Expression (6)). The reward function maps the utility functions to specific observable properties of the simulation platform, which is then used to guide and constrain the behavior of vehicles in the ITS simulation. Specifically, each KAOS goal model k_i contains a number M_i of utility functions f_{ij} associated with leaf-level goals, where $1 \leq j \leq M_i$. Each f_{ij} is a real-valued function (e.g., see Expression (3)) that maps a specific requirement of player p_i's goal model k_i to a satisficement value [22]. Each utility function associated with the leaf-level goals is assigned a weight value $\alpha_{ij}, \alpha_{ij} \in \mathbb{R}$, denoting a developer-specified weight (e.g., to indicate the importance or impact) of the given goal. The reward function π_i directly guides and constrains the behavior of vehicles in the ITS simulation.

$$\pi_i(s_i) = \sum_{j=1}^{M_i} \alpha_{ij} \cdot f_{ij}(p_i). \tag{6}$$

3.4 Step 3. Game-Based Testing

This section overviews how SavviDriver integrates and operationalizes the KAOS goal models and their utility functions into a game to explore human-based uncertainty in an ITS simulation. An `ITS Simulator` is used to keep track of the operating environment's state for a given gaming process (see Figure 2, *ITS Simulator*). For each timestep of a game, the following happens. First, agents use their respective strategies to select actions (e.g., braking, throttle, steering, etc.) which are then provided as input to the `ITS Simulator`. Second, the `ITS Simulator` executes the collective actions from all the participating agents. The `ITS Simulator` then updates the environment (e.g., vehicle positions, velocities, etc.) and returns a new ITS state to the respective gaming process. The sequence of each agent's actions and resulting environment states inform the RL learning process and may be archived (e.g., failure traces) to support further analysis processes. We next overview how SavviDriver uses game-based testing in simulation to discover human-based uncertainty potentially impacting an Ego.

Step 3a. NeV Training

First, SavviDriver trains the NeV(s) to complete driving tasks while exhibiting a specific driving style in the presence of the Ego (provided as input by a developer). The NeV(s) are in training mode, where the DNN is actively learning new behavior in response to its environment. The goal models from **Step 1b** inform the reward structure of each NeV. Thus, the NeV(s) learn to complete their respective functional goal specified in the corresponding KAOS goal model that reflects a particular driving style (e.g., aggressive, distracted, timid, etc.). For example, an aggressive NeV may exploit the safe driving behavior of an Ego (i.e., maintains a conservative following distance) by cutting in front of the Ego in order to satisfy a NeV non-functional goal to get to its destination quickly.

Step 3b. AV Uncertainty Discovery

Next, SavviDriver assesses the ability of the Ego to operate appropriately in the presence of the trained NeV(s) (trained in **Step 3a**). The NeV(s) in *execution mode* choose strategies learned during training (**Step 3a**) according to their reward functions; they do not learn new behavior in response to agent interactions. During this step, the Ego may exhibit previously unseen or unexpected behavior as it responds to the other "human-operated" vehicles exhibiting different driving styles. For example, when the aggressive NeV cuts off the Ego, the (unexpected) Ego response may be to swerve to avoid a collision, which may lead to another collision. The result of this step is a collection of *failure traces*. Failure traces are sequences of agent actions and corresponding environment states (i.e., trace data), indexed by timesteps, for a given execution of the mixed traffic scenario where the Ego did not reach its target destination, perhaps due to a collision. Developers can use failure traces to visualize and replay different executions of mixed traffic scenarios to better understand the interactions and environment states that led to interesting behavior or unexpected Ego failures.

3.5 Step 4. AV Robustification

Similar to simulation-based testing [54, 55, 56], SavviDriver's generated information (i.e., failure traces from **Step 3b**) can be used by developers to revise the system to prevent and/or mitigate the discovered unexpected Ego behavior. The information can be used in several ways, including updating system requirements, modifying system implementation, or used as training data for learning-enabled system components. We next elaborate on these approaches to robustification.

Robustification of Software-Based AVs

For traditional software-based Egos (e.g., Intelligent Driving Model (IDM) [57]), developers can use failure traces to identify goal violations (e.g., collisions, failure to drive towards destination, etc.) and determine how to revise the behavior of the Ego. For example, developers may observe similar failures (i.e., collisions) during merge maneuvers when another vehicle is in close proximity to the Ego, resulting in low rewards. Developers can then use this collection of similar driving patterns/scenarios to specify goals that represent a "cautious" operating mode that increases the desired inter-vehicle distance to avoid collisions during merge maneuvers.

Robustification of Learning-Based AVs

For learning-based Egos, SavviDriver supports robustification by retraining RL-based Egos in the context of the discovered uncertainty (i.e., failure traces from Step 3a-ii). During retraining, the Ego is in training mode and the NeV(s) are in execution mode. The Ego agent learns to complete the goals specified in its KAOS goal model in the presence of uncertainty posed by the NeV(s) that have learned to exhibit specific driving styles specified by their corresponding goal models. The result of this step is an automatically retrained Ego that can move robustly and safely, completing the given task in the presence of human-based uncertainty exhibited by the NeV.

4 Demonstration

To demonstrate the use of the SavviDriver framework to discover unexpected human-based behavior(s) and assess Ego responses, we have applied it in several proof-of-concept use cases based on real-world traffic accident data [23, 58, 59, 60]. We present results for the discovered mixed traffic interactions that cause Ego failure, and then illustrate how developers can use these results to improve an Ego's robustness in response to the human-based uncertainty.[2]

Experiment Setup

Our demonstration is intended as a proof-of-concept study to illustrate the modeling, uncertainty discovery, and robustness analysis capabilities of our framework. To this

[2]A demonstration package is available at https://anonymous.4open.science/r/savvidriver-demo/.

end, we leverage existing tooling for our experiments. First, we use the TinyRoad simulation platform [8] to provide a light-weight 2D traffic environment. TinyRoad uses a bicycle-kinematics physics model [61] and simple 2D graphics that require less computational overhead when compared to more sophisticated and computationally expensive simulation platforms (e.g., CARLA [62], Gazebo [63]). Thus, the selected simulation platform supports quick training and evaluation of RL agents while preserving the key behavioral characteristics of Ego and NeV interactions. Next, we use Proximal Policy Optimization [13] and the OpenAI Gym [64] libraries to implement RL training and agents within the simulation.

Evaluation Metrics

We use two well-established and commonly-used vehicle safety indicators to assess the performance of the Ego: the Average Failure Rate (AFR) [65] and the average Time Exposed Time-to-collision (TET) [66, 67]. The safety metrics are defined over a set of episodes E, where $|E| = 100$ in each use case. An episode is an execution of the mixed traffic simulation from an initial state to a terminal state (i.e., the Ego reaches its destination or a collision occurs).

In our studies, a *failure* is defined as a collision between the Ego under study and an NeV or a road obstacle. Expression (7) defines an indicator function δ_{AFR} that evaluates to 1 if a collision occurs, and 0 otherwise. The AFR metric is defined as the average number of collisions observed over episode set E (see Expression (8)).

$$\delta_{\text{AFR}}(e) = \begin{cases} 0 & \text{if } \textsc{Ego}.\mathit{state} \neq \text{crashed} \\ 1 & \text{if } \textsc{Ego}.\mathit{state} == \text{crashed} \end{cases} \tag{7}$$

$$AFR(E) = 100 \times \frac{1}{|E|} \sum_{e \in E} \delta_{\text{AFR}}(e) \tag{8}$$

The Time-to-Collision (TTC) metric [67] is defined as the time remaining until a collision occurs, where collisions are predicted using linear projection of each vehicle's future position [65]. Expression (9) defines the TTC value for an AV belonging to vehicle set $\mathcal{I}$ at a given timestep t. The distance formula d calculates the predicted Euclidean distance between the position $pos_{\text{AV}}(t + \tilde{t})$ of the AV, and the position $pos_j(t+\tilde{t})$ of vehicle $p_j, p_j \in \mathcal{I} - AV$ at future time $t+\tilde{t}$. When no collision is predicted, the TTC value is set to ∞ [65].

$$\begin{aligned} TTC(\textsc{Ego}, \mathcal{I} - \textsc{Ego}, t) = \min\,(\{\{\tilde{t} \geq 0 \mid d(pos_{\text{AV}}(t+\tilde{t}), \\ pos_j(t+\tilde{t})) = 0\} \cup \{\infty\}\}), \forall j \in \mathcal{I} - \textsc{Ego}. \end{aligned} \tag{9}$$

Next, Expression (10) defines an indicator function δ_{TET} that evaluates to 1 if an input value *ttc* is less than or equal to the given time threshold (seconds) $\tau, \tau \in \mathbb{R}$, and 0 otherwise. Finally, the TET metric is defined as the duration the Ego was exposed to a TTC value below some threshold τ during episode e's time interval $\{0, t_e\}$, averaged over episode set E (see Expression (11)) [65, 66].

$$\delta_{\text{TET}}(ttc) = \begin{cases} 0 & \text{if } ttc > \tau \\ 1 & \text{if } ttc \leq \tau \end{cases} \tag{10}$$

$$TET(E) = \frac{1}{|E|} \sum_{e \in E} \left(\frac{1}{t_e} \sum_{t=0}^{t_e} \delta_{\text{TET}}(TTC(\text{Ego}, \mathcal{I} - \text{Ego}, t)) \right) \tag{11}$$

4.1 Use Case: Left Turn

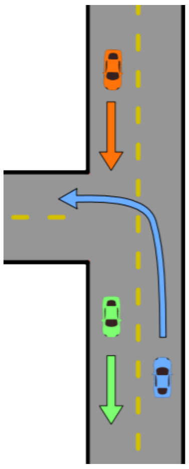
Our first use case explores an uncontrolled left turn intersection. According to the NHTSA [68], a left-turn maneuver is one of the most frequent pre-crash events, especially at intersections without controlled signals [58, 59]. Figure 7a provides an overview of the left-turn use case considered, where the blue Ego seeks to make a left turn and the orange Aggressive-NeV and green Distracted-NeV are traveling in the left lane (towards the blue Ego). Figure 7b provides the corresponding goal model for the green Distracted-NeV.

Data

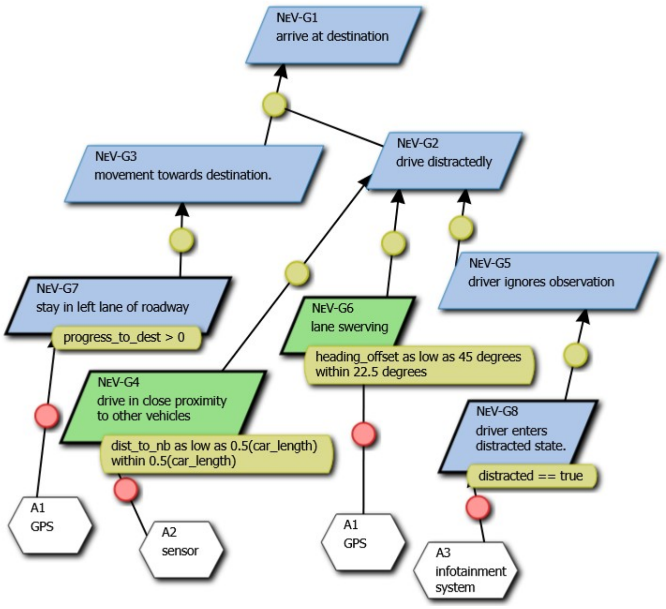
This section describes the key information that is used to define the scenario for this use case. The *scenario* is an unprotected left-turn intersection that comprises a two-lane road with a perpendicular road connecting to the left lane. Therefore, a vehicle in the right lane attempting to make a left turn must cross a lane with oncoming driving. The *agents* participating in this scenario are an aggressive driver (orange), a distracted driver (green), and the AV (blue). The AV under study uses a traditional IDM for decision-making. The *road layout* is modeled based on real-world map data, and we provide the road geometry as input to SavviDriver.

Step 1. Modeling

This section describes the development of goals models (**Step 1**) for our left-turn use case. We explore two different NeVs, an Aggressive-NeV and a Distracted-NeV operating in a multi-agent setting. For the Aggressive-NeV, we reference the component library to reuse the non-functional subtree of the Aggressive-NeV defined in our merge running example (see Section 3). The previously developed non-functional subtree is based on NHTSA's traffic data that identifies three leading causes of accidents related to aggressive drivers: speeding, proximity to other road users, and inability to center in a lane [17, 18, 19]. We update the functional objective of the Aggressive-NeV to reflect the functional context-dependent goal: NeV-G7 '`stay in right lane of roadway`'. This process demonstrates how SavviDriver facilitates the reusability of goal models by using previously defined agent goal(s) in a new scenario. Figure 7b shows the KAOS model for the Distracted-NeV, where the top-level goal of NeV-G1 '`arrive at destination`' is decomposed into subgoals such as NeV-G8 '`driver enters distracted state`', NeV-G6 '`lane swerving`', and NeV-G4 '`drive in close proximity to other vehicles`'.



(a) Graphical depiction of left-turn use case.


NEV-G1
arrive at destination
NEV-G3
movement towards destination.
NEV-G2
drive distractedly
NEV-G5
driver ignores observation
NEV-G7
stay in left lane of roadway
progress_to_dest > 0
NEV-G6
lane swerving
heading_offset as low as 45 degrees within 22.5 degrees
NEV-G4
drive in close proximity to other vehicles
dist_to_nb as low as 0.5(car_length) within 0.5(car_length)
NEV-G8
driver enters distracted state.
distracted == true
A1
GPS
A2
sensor
A1
GPS
A3
infotainment system

(b) KAOS goal model for the green DISTRACTED-NEV.

Fig. 7: Overview of left-turn use case and green DISTRACTED-NEV goal model. The left-turn use case involves three vehicles. The blue EGO attempts to make a left turn, while the orange AGGRESSIVE-NEV and green DISTRACTED-NEV are driving in the left lane.

Step 2. Middleware

This section describes how SAVVIDRIVER processes goal models and scenario information to automatically initialize the gaming setup and simulator for the left-turn use case. First, **Step 2a** takes as input the scenario definition file that captures key context-dependent information for this use case and instantiates the simulator. Next, the reward function for each agent is automatically compiled from the aggregate utility functions in their respective goal models (**Step 2b**). Listing 2 provides a pseudocode example of the automatically generated reward function for the DISTRACTED-NEV. The utility function associated with goal NEV-G4 rewards the DISTRACTED-NEV for driving as close as possible to other vehicles, with a maximum reward assigned when its distance to the closest vehicle is less than half of its length. The utility function associated with NEV-G6 rewards the DISTRACTED-NEV for swerving and is parametrized by the current heading offset from the lane center. The maximum reward is achieved when the current offset is greater than 45 degrees. The utility function associated with NEV-G8 rewards the DISTRACTED-NEV for entering a distracted state and is parametrized by a boolean value that indicates if the driver is distracted. The NeV-G8 utility function returns a reward of 1 if the driver is distracted and 0 otherwise. The NeV-G7 rewards the Distracted-NeV for making progress between timesteps. The NeV-G7 utility function returns a reward of 1 if the driver makes progress and 0 otherwise. Each utility function evaluates to a real-valued number (i.e., level of satisficement). Using Expression (6), the reward is computed as the linear weighted sum of the satisficement values multiplied by developer-specified weights. Since the Aggressive-NeV has the same non-functional objectives as the merge running example, we regenerate the Aggressive-NeV's reward structure to reflect the new context-dependent functional objective for the left-turn use case. We describe the results for the left-turn use case next.

Listing 2: Pseudocode for the Distracted-NeV's reward function corresponding to its KAOS goal model.

```
def reward(p):
    # utility function for G4
    #   p.nb := distance to nearest neighbor
    #   p.s  := car length of NeV
    fi_0 = (0 if p.nb >= 0.5*p.s else
        (1 if p.nb < 0.5*p.s else
        (0.5*p.s-p.nb)/0.5))

    # utility function for G6
    #   p.hof := NeV's heading offset from current lane
    fi_1 = (0 if p.hof <= 22.5 else
        (1 if p.hof >= 45 else (45-p.hof)/22.5))

    # utility function for G8
    #   p.d := boolean indicating if NeV is distracted
    fi_2 = (0 if not p.d else 1)

    # utility function for G7
    #   p.prog := percent increase in route completion from last update
    fi_3 = (1 if p.prog > 0 else 0)

    # weights - default is equal weights
    alpha = [1, 1, 1, 1]

    # Reward function (see Expression (4))
    uf = [fi_0, fi_1, fi_2, fi_3]
    reward = 0
    for j in range(|uf|):
        reward += alpha[j]*uf[j]
    return reward
```

Step 3. Game-Based Testing

Figure 8 illustrates a failure scenario observed during the uncertainty discovery process (**Step 3b**) for the left-turn use case. The blue Ego's initial attempt to make a left-turn maneuver (see Ⓐ, Figure 8) is aborted due to the swerving behavior of the green Distracted-NeV. Then the blue Ego attempts another left-turn maneuver in close proximity to the orange Aggressive-NeV, causing it to enter the Ego's lane (see Ⓑ, Figure 8). The orange Aggressive-NeV's behavior causes the blue Ego to accelerate and collide with the road barrier (see Ⓒ, Figure 8), therefore resulting in an increased failure rate when the two NeVs are in close proximity to the Ego's lane. Table 2 provides quantitative results for the left-turn use case. In the absence of the NeVs, the Ego is able to complete the left turn with a failure rate of approximately 0.00%. However, when exposed to the distracted and aggressive NeVs, the Ego's failure rate increases to 12.00%. Additionally, the observed TET values in Table 2 indicate that the Ego was exposed for the highest average duration of time to critical TTC values τ, $\forall \tau \in \{1, 2, 3\}$ in the presence of the NeVs. TTC values below a specific time threshold (e.g., $\tau = 3$ seconds [66]) potentially indicate that a vehicle would not have enough time to perform a collision-avoiding maneuver, such as applying the brakes to stop the vehicle. We observe higher average TET values indicating the Ego was in a near-collision state when exposed to both NeVs.

Step 4. Robustification

The discovered uncertainty and corresponding failure traces are used to inform a reconfiguration of the Ego's IDM, a rule-based controller. Specifically, during the replay and analysis of the failure traces, we observed that the Ego did not properly wait for an opening to make a left turn, attempting the turning maneuver in close proximity to other vehicles. We confirmed our visual observations through further examination of the failure trace data, specifically the distance between the vehicles leading up to a failure. Informed by our analysis, we developed a KAOS functional subtree for 'cautious' mode [69] for the Ego that implemented a blocking routine until the intersection is clear with respect to a given distance threshold before attempting to perform the left-turn maneuver. After reconfiguration in **Step 4**, the robustified Ego is better able to wait for openings in the intersection, thereby reducing its failure rate from 12.00% down to 1.00%. This use case demonstrated how SavviDriver was able to reuse existing goal models, extend them with different numbers and types of drivers, apply them to a new ITS use case to discover unexpected human behavior with the corresponding unexpected Ego responses, and then use this information to reconfigure the Ego, resulting in a reduced failure rate.

Table 2: Evaluation of *AFR* and *TET* metrics for the left-turn case study, where red color indicates the worst value for each metric. A higher AFR indicates a higher average collision rate and a higher TET indicates higher average time spent in a near-collision state (i.e., predicted time-to-collision less than τ seconds).

Setting	**AFR**	**TET**		
		$\tau = 1$s	$\tau = 2$s	$\tau = 3$s
Base AV	0.00%	0.00%	0.00%	0.50%
Uncertainty Discovery	**12.00%**	**2.90%**	**3.80%**	**4.90%**
Robustified AV	1.00%	0.10%	0.80%	2.70%

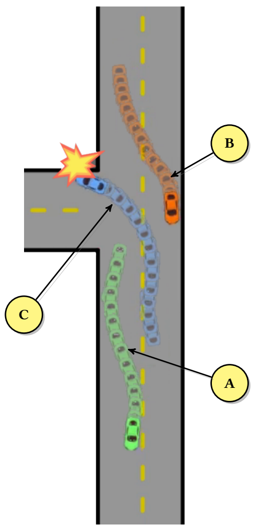


Fig. 8: Left turn use case overview. The green DISTRACTED-NEV's sporadic movement led to a preemptive turn in the blue EGO. The EGO then speeds up to avoid collision with the orange AGGRESSIVE-NEV, resulting in a crash of the EGO.

4.2 Use Case: Weave Lane

Our next use case explores a weave lane highway merge. Figure 9 provides a graphical depiction of the weave lane highway merge use case, where the blue EGO is trailed by the orange NEV. Both vehicles must exit the on-ramp and merge onto the highway. A weave lane is a challenging highway merge scenario, as vehicles entering the highway often start at a low speed due to the entrance curve speed limit and can cause automated vehicles to fault [23]. For example, a 2020 Waymo EGO exhibited unexpected behavior during a highway merge scenario where the EGO failed to complete the merge maneuver [60]. We next detail how SAVVIDRIVER is used to model and explore specific human-based behaviors that may lead to vehicle failures in the weave lane scenario.

Data

Using historical data and/or domain expertise, we identify three types of information required for SAVVIDRIVER and capture them in a scenario definition. The *scenario* is a weave lane that comprises a highway on-ramp that merges onto the main lane of the highway. The *agents* participating in this scenario are the AGGRESSIVE-NEV and the EGO. In contrast to the left-turn use case, this use case explores an EGO that uses an RL-based controller for decision-making that was trained to satisfy the requirements captured in the goal model shown in Figure 6a [8]. The *road layout* is modeled based on real-world map data, and we provide the road geometry as input to SAVVIDRIVER.

Step 1. Modeling

This section describes the development of goal models for the weave lane use case. Since the non-functional objectives describe *how* an agent should achieve a task, they can often be reused between different driving scenarios. In this use case, we reuse the non-functional subtrees saved in our `Component Library` for both the Aggressive-NeV and Ego. For both agents, we update their functional objective to reflect the context-dependent top-level function goal: '`merge onto highway`'. This process illustrates how a developer may leverage the `Component Library` during **Step 1** to reuse existing context-independent non-functional subtrees and apply them in a new scenario to discover additional sources of human-based uncertainty.

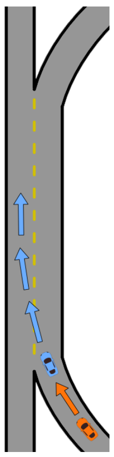


Fig. 9: A scenario capturing the weave lane use case in TinyRoad. Both the Ego and Aggressive-NeV are traveling from an on-ramp and attempt to merge onto the highway.

Step 2. Middleware

In this step, SavviDriver processes goal models and scenario information to automatically initialize the gaming setup and simulator for the weave lane use case. **Step 2a** processes the scenario definition file and instantiates the simulator. Next, the goal models are automatically processed to generate the reward structure for the game-based testing process (**Step 2b**). Specifically, utility functions associated with leaf-level goals are aggregated into a linear weighted sum that represents the reward for each agent in the game. Notably, SavviDriver automatically generates a new reward structure for the weave lane use case through model-level updates rather than low-level simulation code changes.

Step 3. Game-Based Testing

Figure 10 illustrates an Ego failure observed during the uncertainty discovery phase. In the presence of the NeV, the Ego attempted an early merge maneuver onto the highway to maintain a safe distance away from the Aggressive-NeV. However, the Ego failed to account for the trajectory of the NeV, resulting in an increased failure rate when both vehicles merged in close proximity to each other. Table 3 shows a comparison between the average Ego AFR and TET over 100 random episodes. In this use case, we use an AV that has been trained with an RL as a baseline. The baseline Ego is able to complete the highway merge in the presence of external rule-following traffic with a failure rate of only 2.0%. However, when exposed to the aggressive NeV (**Step 3b**), the Ego's failure rate increases to 34.0%. Additionally, the observed TET values (see Table 3) indicate that the Ego was exposed for the highest average duration of time to critical TTC values (i.e., τ, $\forall \tau \in \{1, 2, 3\}$) in the presence of the aggressive NeV. Higher average TET values indicate the Ego was more frequently exposed to near-collision scenarios, which may contribute to the increased failure rate. For example, we observe that the Ego was in a near-collision state (within $\tau = 3$ seconds) for 12.60% of the episodes when exposed to the aggressive NeV.

Step 4. Robustification

Step 4 robustifies the Ego vehicle by retraining its RL-based controller in the presence of the discovered uncertainty. After retraining, the Ego learned to reduce its speed during the highway merge maneuver, thereby reducing its failure rate to 18.0% (down from 34.0%). Specifically, after retraining, the Ego learned to delay the merging maneuver to allow the aggressive driver to pass at a safe distance. This demonstration also illustrated how SavviDriver was able to reuse composable components of existing behavior models and apply them to a new use case to discover dangerous human behavior and unexpected Ego responses. Then SavviDriver used this information to retrain the Ego, resulting in a reduced failure rate.

Table 3: Evaluation of *AFR* and *TET* metrics for the weave lane case study, where red color indicates the worst value for each metric. A higher AFR indicates a higher average collision rate and a higher TET indicates higher average time spent in a near-collision state (i.e., predicted time-to-collision less than τ seconds).

Setting	**AFR**	**TET**		
		$\tau = 1s$	$\tau = 2s$	$\tau = 3s$
Base AV	2.00%	4.20%	7.50%	7.90%
Uncertainty Discovery	**34.00%**	**8.60%**	**11.40%**	**12.60%**
Robustified AV	18.00%	5.00%	7.10%	8.00%

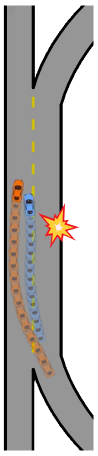


Fig. 10: Demonstration of a failure discovered by SavviDriver, where the orange Aggressive-NeV completes its merge before the blue Ego and speeds up, causing a collision as the Ego tries to merge.

4.3 Use Case: Adaptive Cruise Control

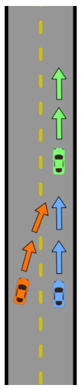
Our final use case explores a merge scenario involving an AV equipped with a traditional software implementation of an Adaptive Cruiser Control (ACC) system, an Advanced Driver Assistant system (ADAS) feature found in most modern vehicles. The ACC system uses onboard sensors, such as radar and lidar, to maintain a safe trailing distance between itself and a *target* vehicle. The goal of this use case is to illustrate how SavviDriver may be used to assist in the engineering, assessment, and manual reconfiguration of ADASs (i.e., ACC) by developers. Figure 11a shows the road and vehicle configuration for the ACC scenario. An orange NeV seeks to merge into the lane of the blue Ego AV before reaching the end of the road. The AV seeks to maintain a safe trailing distance to a green target vehicle traveling at a constant speed.

Data

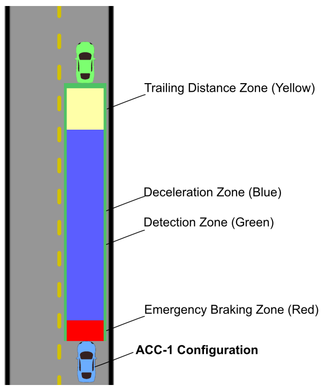
This section describes the key information that is used to define the scenario for this use case. The scenario is a highway road that comprises two lanes traveling in the same direction. The agents participating in this scenario are an aggressive driver, a target vehicle, and the AV. The road layout is modeled based on real-world map data, and we provide the road geometry as input to SavviDriver.

Step 1. Modeling

The ACC-1 software is based on real-world Original Equipment Manufacturers' (OEM) ACC implementations [70, 71]. Figure 12 shows a sample goal model corresponding to the ACC system [14]. The initial ACC configuration (ACC-1) is shown in Figure 11b and implemented as follows. If no target vehicle is detected in the detection zone (i.e., green rectangle), then ACC-1 accelerates to maintain the set speed. If a target vehicle is detected, then the ego vehicle constantly decelerates over a large distance (i.e., blue rectangle) and maintains the same speed as the target vehicle in the trailing distance zone (i.e., yellow rectangle). Additionally, the ACC-1 controller is equipped with an Emergency Braking System (EBS). The EBS applies a hard brake (i.e., maximum braking force) when a target vehicle is detected within the emergency braking zone of the ego vehicle (i.e., red rectangle). For the AGGRESSIVE-NEV, we reuse the goal model described in our running merge example in Section 3 (see Figure 6b). This agent's functional goal is to merge into the EGO's lane before they reach the end of the road.



(a) Overview of the ACC use case.



(b) Graphical depiction contrasting the ACC-1 system.

Fig. 11: Overview of the ACC use case and the ACC-1 system. Subfigure (a) shows the ACC use case that involves three vehicles. The green NeV target vehicle travels straight at a constant speed. The blue EGO vehicle uses a rule-based ACC controller to maintain a fixed trailing distance to the vehicle in front. The orange AGGRESSIVE-NEV seeks to merge into the right lane. For Subfigure (b), the yellow box shows the trailing distance the ego vehicle aims to maintain. The green box denotes the distance when the ego vehicle detects the green non-ego target vehicle. The blue box indicates the deceleration zone. Finally, the red box denotes the emergency braking zone, where the ego vehicle applies the maximum braking force if an object is detected within the zone.

Step 2. Middleware

Similar to the previous use cases, SAVVIDRIVER automatically instantiates the simulation environment, gaming setup, and compiles the reward function for each agent using the aggregate utility functions from their respective goal models. We next describe the game-based testing and discovered uncertainty for the ACC use case.

Step 3. Game-Based Testing

Figure 13a illustrates a critical scenario observed during uncertainty discovery. The orange AGGRESSIVE-NEV suddenly merges in front of the blue EGO, causing the ACC system to activate its EBS. Table 4 shows a comparison between the average duration(s) of EBS activation over 100 random episodes. In the baseline scenario, the Ego does not activate its EBS as it follows the target vehicle. However, when exposed to the Aggressive-NeV, the average EBS duration increases from 0.00 seconds to 3.52 seconds. This increase indicates the ACC experienced more near-collision events in the presence of the Aggressive-NeV.

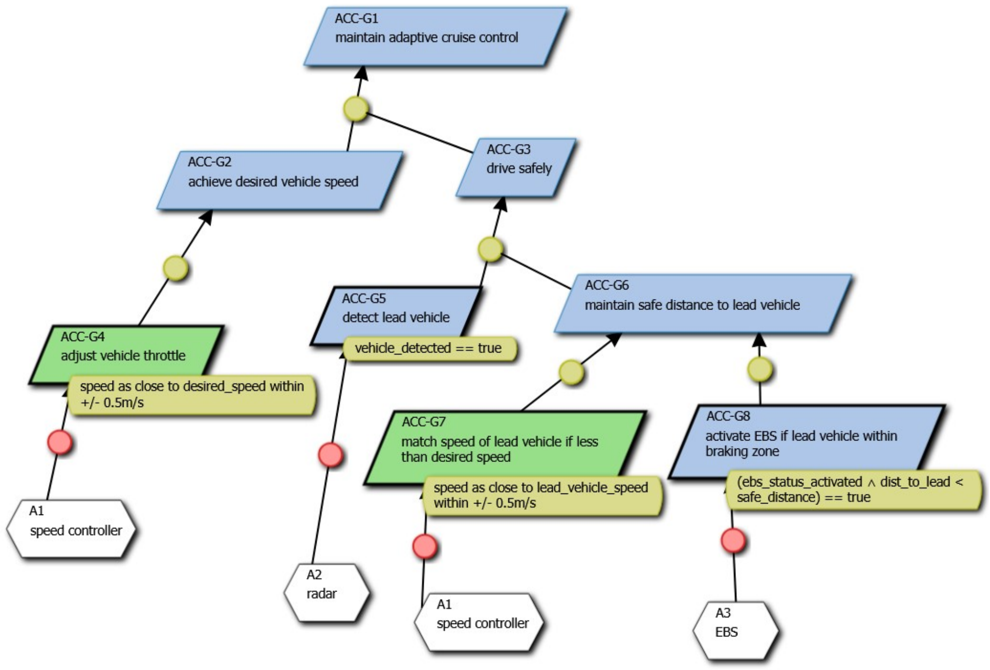
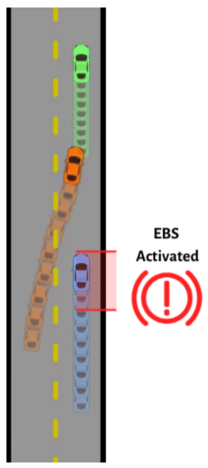

ACC-G1
maintain adaptive cruise control
ACC-G2
achieve desired vehicle speed
ACC-G3
drive safely
ACC-G5
detect lead vehicle
vehicle_detected == true
ACC-G6
maintain safe distance to lead vehicle
ACC-G4
adjust vehicle throttle
speed as close to desired_speed within +/- 0.5m/s
ACC-G7
match speed of lead vehicle if less than desired speed
speed as close to lead_vehicle_speed within +/- 0.5m/s
ACC-G8
activate EBS if lead vehicle within braking zone
(ebs_status_activated ∧ dist_to_lead < safe_distance) == true
A1
speed controller
A2
radar
A1
speed controller
A3
EBS

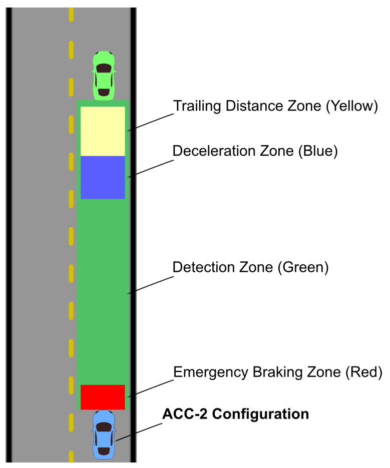
Fig. 12: KAOS model for ACC subsystem.

Step 4. Robustification

Observing the unexpected maneuver from the Aggressive-NeV and response from the Ego configured with ACC-1, a developer can engineer or reconfigure the ACC system, yielding an alternative ACC-2 controller to mitigate the unsafe scenario in **Step 4**. Figure 13b shows an overview of the ACC-2 configuration. In contrast to the gradual deceleration when a target vehicle is detected by the ACC-1 controller (see Figure 11b), the ACC-2 controller's behavior is revised to maintain its speed until it reaches a short deceleration zone before applying the brakes to match the speed of the target vehicle (see Figure 13b). After robustification, the average duration of EBS activation over 100 episodes decreases to 0.16 seconds in the presence of the Aggressive-NeV. As such, this use case highlights how a developer may use SavviDriver to identify problems in an initial configuration of an ACC controller, engineer a mitigating solution, and confirm the solution addresses previously identified unexpected behaviors. The reconfiguration of the ACC system demonstrated in this use case captures a real-world use case when ACC systems were initially deployed in the early 2000s by OEMs.[3] First iterations of ACC systems applied a similar configuration as our ACC-1 system. As OEMs discovered that aggressive drivers may cut off the ADAS-equipped vehicles, they changed the behavior of their ACC models to apply the brakes as it enters the "short deceleration zone" of the target vehicle (i.e., ACC-2). This example shows that by applying SavviDriver, developers can identify unsafe situations during testing and modify the behavior of their system to mitigate similar situations.



(a) Example trajectory of the orange Aggressive-NeV who abruptly merged in front of the blue Ego, activating the EBS of the Ego.



(b) Graphical depiction contrasting the ACC-2 system. This configuration maintains a smaller trailing distance compared to ACC-1.

Fig. 13: Overview of (a) sample uncertainty discovered by SavviDriver and (b) the corresponding robustified Ego configuration.

Table 4: Evaluation of ACC use case, where red color indicates the worst value for the EBS duration metric. A higher EBS duration indicates a higher average time spent in a near-collision state.

Setting	**EBS Duration (s)**
Base AV (ACC-1)	0.00
Uncertainty Discovery	**3.52**
Robustified AV (ACC-2)	0.16

[3]This information is based on data provided by our industrial collaborators.

5 Related Work

This section overviews related work relevant to SavviDriver. First, we discuss related work with game theory and RL. Next, we discuss relevant research for game-based testing of AVs. Finally, we overview other model-based approaches to RL, uncertainty discovery, or AVs.

A number of researchers have proposed game theory or RL-based approaches to train AV logic or improve the robustness of AVs against uncertainty. Liniger *et al.* [38] proposed a non-cooperative game theory approach to train agents for autonomous racing games, but uses the same reward objective for all agents (i.e., agents compete towards the same goal). Cao *et al.* [72] proposed an imitation learning approach to learn different types of driving models for an AV, and uses RL to learn to swap between the different learned policies. Li *et al.* [73] proposed a cooperative game-theoretic approach to model driver and vehicle interactions, but do not consider human-based uncertainty using a non-cooperative setting and have strictly defined reward functions for RL. Gupta *et al.* [74] introduced a framework to train two dueling agents in an RL setting, training a resilient ego vehicle against a (possibly adversarial) non-ego agent. Zhou *et al.* [75] proposed the UBRL framework, identifying potentially unreliable decisions of an RL agent, but does not account for human-induced uncertainty. Chan *et al.* [8] demonstrated that RL and non-cooperative game theory can be combined to discover undesirable AV behaviors. However, existing works largely use ad hoc development approaches to structure game objectives and/or traffic scenarios. Our work uses goal models to explicitly capture the functional/non-functional objectives of different driving styles and provides a reusable approach to explore human-based uncertainty for AVs through gaming and RL.

Game-based testing has been used to assess and/or improve AVs with respect to uncertainty posed by other drivers. Weiwei *et al.* [12] propose a multi-agent RL framework to train AVs in the presence of other vehicles, where different vehicle behavior is realized by adjusting the proportion between cooperative and selfish rewards. Hao *et al.* [9] use adversarial games and RL to test AV robustness to vehicles that are trained to cause AV failures (e.g., collisions) while exhibiting behaviors that mimic real-world driving action distributions (i.e., naturalistic priors). Wachi *et al.* [11] combine multi-agent games and adversarial RL [37] to discover failure cases for rule-based vehicles in simulation. Ma *et al.* [10] use level-k game theory [76] and RL to assess and improve an IDM decision-making mechanism (i.e., lane changing) in the presence of vehicles with manually defined reward functions that can exhibit cooperative or competitive behavior. However, game-based testing with RL approaches is often tightly coupled to specific types of vehicle models or a single dimension of human-based uncertainty and use ad-hoc development approaches for exploring different traffic scenarios. In contrast, our work provides a modular framework that uses goal models to declaratively specify human-based driving styles and supports the composition of these models to explore multiple dimensions of human-based uncertainty (i.e., different types and/or combinations of driving styles characterized by functional/non-functional objectives).

Other related work has explored testing AV behaviors in simulation. Dosovitskiy *et al.* [62], Son *et al.* [32], and Zhou *et al.* [33] proposed a number of simulation environments that can be used for AV testing. Birchler *et al.* [77], Zheng *et al.* [78], Zhong *et al.* [79] proposed several methods to generate test cases for an AV in simulation, but do not consider uncertainty from human-induced behaviors. Gambi *et al.* [34] combined procedural content generation and search-based testing to explore AV failure in various automatically generated traffic scenario settings. Stocco *et al.* [80] introduced ThirdEye, a white-box AV failure predictor using machine learning that periodically monitors the AV's reliability by identifying instances where the AV may be unconfident. While addressing robustness, they do not address human-induced uncertainty nor support a model-based approach. Fremont *et al.* [81] proposed Scenic, a probabilistic modeling language for specifying and generating a distribution of simulation environments for testing AVs. However, their approach focuses on modeling the environment and does not address means to capture non-functional objectives of agents in the environment.

Other researchers have proposed model-based approaches for AVs or RL. Langford *et al.* [22] introduced MoDALAS, demonstrating a model-based approach to manage and verify the run-time assurance of machine learning components using KAOS goal models and utility functions. However, their approach does not consider human-based non-functional objectives when addressing run-time assurance. Liaskos *et al.* [82] proposed extending the i* framework to model a formal specification for Markov decision processes. Rudolph *et al.* [83] proposed a method to identify and consider strategies of RL agents in the presence of other players for self-adaptation. Jiang *et al.* [84] proposed the THGC model, which uses prior domain knowledge to group agents in a multi-agent reinforcement learning setting. Bruggner *et al.* [85] proposed a model-based approach to AV simulation, but models the different components of individual autonomous agents (i.e., perception, planning, and control). Leung *et al.* [86] introduced goal modeling for RL agents, where policies are represented as goals. However, their work does not use models to capture the high-level objectives of the agents and their interactions. Schwan *et al.* [41] proposed a goal specification language to formalize a reward function for a given RL task (e.g., drive to destination). In contrast, our approach uses KAOS-inspired goal models to specify RL rewards as a means to train agents that exhibit specific driving styles, including those that are human-based.

Finally, researchers have also proposed techniques that consider the role of humans during the development and deployment of cyber-physical systems (e.g., AVs). Cleland-Huang *et al.* [87] and Camara *et al.* [88] focused on humans as an *internal* and *collaborative* contributor to provide decision-making support and other self-adaptive actions (e.g., MAPE-K actions [89]). Orthogonally, we focus on assessing/improving the robustness of AVs with respect to *external* human-based uncertainty in a *non-cooperative* setting. Gavidia-Calderon *et al.* [90] focused on the detection of different types of humans in the adaptive system's environment, which then informs system adaptations. Our objective, in contrast, is to discover human-based uncertainty-induced *edge cases* that are detrimental to the safe operation of AVs.

6 Threats to Validity

This paper described an agent-based goal modeling approach to uncertainty discovery for AVs using non-cooperative game theory and RL. There may be possible deviations between agent behaviors observed in simulation and reality (i.e., "reality gap" [91]). The relevance of the discovered behavior is limited by how well the goal models capture the intended driving styles of vehicles in mixed-traffic interactions. The goal models used in this work were designed by domain experts in the field of automotive assurance, and variations to the KAOS goal models may yield different experimental results [92]. Finally, repeated experiments may lead to or discover different types of agent behaviors or uncertainty, as RL uses stochastic processes to train agents. To ensure the feasibility of the approach, each use case shows the result of an average of 100 episodes for comparison, similar to validation studies in other existing game-based testing approaches [9, 10].

7 Conclusion

This paper introduced the SavviDriver framework for using goal modeling to provide a systematic process in the RL-based realization of multi-agent game-based testing to discover unexpected behaviors. SavviDriver is a game-based testing framework that assesses the ability of the AV system to safely interact with various types of human drivers, who may exhibit a range of non-functional driving styles as they achieve their functional goals. Our framework promotes the discovery of interesting and/or unexpected behavior for both the Ego and the NeV(s), which can then be used to assess and improve the robustness of the AV with respect to multiple dimensions of human-based uncertainty.

We applied SavviDriver to three use cases to demonstrate the systematic refinement of high-level human driving styles and objectives into functional and non-functional goals, based on real-world traffic accident data [17, 93]. We implemented the multi-agent game-based testing using RL, based on the reusable goal models, and then demonstrated our framework's ability to discover undesirable behavior in the AV under study. These failures are often caused by selfish, yet non-malicious objectives of the non-ego vehicles. We show that these use cases can discover failure in common challenging road scenes based on existing data from NHTSA and other traffic reports. For the ACC system use case, we were able to use SavviDriver to discover undesirable behaviors, where the failure traces led to a similar set of changes recommended by the OEM during the early deployment stages of the ACC systems [70, 71]

Future work will explore the training of multiple RL agents with additional driving styles (e.g., timid, intoxicated, etc.) and their composition(s) within a given agent to discover unique strategies of agents and potentially reduce training time. Additional studies may explore how evolutionary computation can be used to explore different gradients and/or combinations of human driving styles (i.e., different levels of aggressiveness or speeding). Other research may investigate the use of procedural content generation to automatically create traffic scenarios for study, including those with various environmental conditions (e.g., slippery road surfaces, road obstacles, potholes, etc.).

Acknowledgments

We greatly appreciate contributions from Nick Polanco on our preliminary work. We also thank Joshua E. Siegel and Shaunak D. Bopardikar for the insightful discussions and detailed feedback on our framework. This work was supported in part by funding provided by Michigan State University.

References

[1] Chen, B., Sun, D., Zhou, J., Wong, W. & Ding, Z. A future intelligent traffic system with mixed autonomous vehicles and human-driven vehicles. *Information Sciences* **529**, 59–72 (2020).

[2] Wachenfeld, W. & Winner, H. The release of autonomous vehicles. *Autonomous Driving: Technical, Legal and Social Aspects* 425–449 (2016).

[3] Chao, Q. *et al.* A survey on visual traffic simulation: Models, evaluations, and applications in autonomous driving (2020).

[4] Nash, J. Non-cooperative games. *Annals of Mathematics* **54**, 286–295 (1951). URL http://www.jstor.org/stable/1969529.

[5] Rushby, J. Logic and Epistemology in Safety Cases (2013).

[6] Hüllermeier, E. & Waegeman, W. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning* **110**, 457–506 (2021).

[7] Ramirez, A. J., Jensen, A. C. & Cheng, B. H. C. A taxonomy of uncertainty for dynamically adaptive systems (2012).

[8] Chan, K. H., Zilberman, S., Polanco, N., Siegel, J. E. & Cheng, B. H. C. SafeDriveRL: Combining Non-cooperative Game Theory with Reinforcement Learning to Explore and Mitigate Human-based Uncertainty for Autonomous Vehicles (2024). Short Paper.

[9] Hao, K. *et al.* Adversarial Safety-Critical Scenario Generation using Naturalistic Human Driving Priors. *IEEE Transactions on Intelligent Vehicles* 1–16 (2023).

[10] Ma, Y. *et al.* Evolving testing scenario generation and intelligence evaluation for automated vehicles. *Transportation Research Part C: Emerging Technologies* **163**, 104620 (2024).

[11] Wachi, A. Failure-Scenario Maker for Rule-Based Agent using Multi-agent Adversarial Reinforcement Learning and its Application to Autonomous Driving (2019). arXiv:1903.10654.

[12] Weiwei, L. *et al.* Learning to Model Diverse Driving Behaviors in Highly Interactive Autonomous Driving Scenarios with Multi-Agent Reinforcement Learning (2024). arXiv:2402.13481.

[13] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal Policy Optimization Algorithms (2017). arxiv:arXiv:1707.06347.

[14] Ramirez, A. J., Jensen, A. C., Cheng, B. H. & Knoester, D. B. Automatically exploring how uncertainty impacts goal satisfaction (2011).

[15] Dijkstra, E. W. On the role of scientific thought. *Selected writings on computing: a personal perspective* 60–66 (1982).

[16] Parnas, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* **15**, 1053–1058 (1972).

[17] Media, NHTSA. Driving behaviors reported for drivers and motorcycle operators involved in fatal crashes. https://www.iii.org/fact-statistic/facts-statistics-aggressive-driving (2021).

[18] Media, NHTSA. Nhtsa estimates for 2022 show roadway fatalities remain flat after two years of dramatic increases. https://www.nhtsa.gov/press-releases/traffic-crash-death-estimates-2022 (2023).

[19] Richard, C. M., Campbell, J. L., Brown, J. L. *et al.* Task analysis of intersection driving scenarios: Information processing bottlenecks. Tech. Rep., Turner-Fairbank Highway Research Center (2006).

[20] van Lamsweerde, A. Goal-oriented requirements engineering: A guided tour (2001).

[21] Walsh, W. E., Tesauro, G., Kephart, J. O. & Das, R. Utility functions in autonomic systems (2004).

[22] Langford, M. A., Chan, K. H., Fleck, J. E., McKinley, P. K. & Cheng, B. H. C. Modalas: Model-driven assurance for learning-enabled autonomous systems (2021).

[23] Thwarted on the On-ramp: Waymo Driverless Car Doesn't Feel the Urge to Merge. https://www.thetruthaboutcars.com/2018/05/thwarted-ramp-waymo-driverless-car-doesnt-feel-urge-merge/ (2018).

[24] Bradley, R. Tesla Autopilot. https://www.technologyreview.com/technology/tesla-autopilot/.

[25] Freedman, I. G., Kim, E. & Muennig, P. A. Autonomous vehicles are cost-effective when used as taxis. *Injury epidemiology* **5**, 1–8 (2018).

[26] Us, Y. Overcoming deployment hurdles: Adastec's approach to automated bus integration (2023). URL https://www.adastec.com/post/overcoming-deployment-hurdles.

[27] Team, T. W. Waymo significantly outperforms comparable human benchmarks over 7+ million miles of rider-only driving. https://waymo.com/blog/2023/12/waymo-significantly-outperforms-comparable-human-benchmarks-over-7-million.

[28] Koopman, P., Osyk, B. & Weast, J. Autonomous vehicles meet the physical world: Rss, variability, uncertainty, and proving safety (2019).

[29] Sagberg, F., Selpi, Bianchi Piccinini, G. F. & Engström, J. A review of research on driving styles and road safety. *Human factors* **57**, 1248–1275 (2015).

[30] Paleti, R., Eluru, N. & Bhat, C. R. Examining the influence of aggressive driving behavior on driver injury severity in traffic crashes. *Accident Analysis & Prevention* **42**, 1839–1854 (2010).

[31] Antić, B., Čabarkapa, M., Čubranić-Dobrodolac, M. & Čičević, S. The influence of aggressive driving behavior and impulsiveness on traffic accidents. *US Transportation Collection* (2018).

[32] Son, T. D., Bhave, A. & Van der Auweraer, H. Simulation-based testing framework for autonomous driving development (2019).

[33] Zhou, L. *et al.* Garchingsim: An autonomous driving simulator with photorealistic

scenes and minimalist workflow (2023).

[34] Gambi, A., Mueller, M. & Fraser, G. Automatically testing self-driving cars with search-based procedural content generation (2019).

[35] Leventhal, L. M., Teasley, B. M., Rohlman, D. S. & Instone, K. Positive test bias in software testing among professionals: A review (1993).

[36] Rapoport, A. *Game theory as a theory of conflict resolution* Vol. 2 (Springer Science & Business Media, 2012).

[37] Pinto, L., Davidson, J., Sukthankar, R. & Gupta, A. Robust adversarial reinforcement learning (2017).

[38] Liniger, A. & Lygeros, J. A noncooperative game approach to autonomous racing. *IEEE Transactions on Control Systems Technology* **28**, 884–897 (2019).

[39] Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* **34**, 26–38 (2017).

[40] Kiran, B. R. *et al.* Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems* **23**, 4909–4926 (2021).

[41] Schwan, S., Klös, V. & Glesner, S. A goal-oriented specification language for reinforcement learning (2023).

[42] Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research* **4**, 237–285 (1996).

[43] Tzorakoleftherakis, E. Reinforcement Learning: A Brief Guide. https://www.mathworks.com/company/technical-articles/reinforcement-learning-a-brief-guide.html (2019).

[44] Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B. & Steenkiste, P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* **37**, 46–54 (2004).

[45] Ramirez, A. J. & Cheng, B. H.C.. Automatic derivation of utility functions for monitoring software requirements (2011).

[46] DeGrandis, P. & Valetto, G. Elicitation and utilization of application-level utility functions (2009).

[47] Fredericks, E. M. Automatically hardening a self-adaptive system against uncertainty (2016). URL http://doi.org/10.1145/2897053.2897059.

[48] Mergia, W. Y., Eustace, D., Chimba, D. & Qumsiyeh, M. Exploring factors contributing to injury severity at freeway merging and diverging locations in ohio. *Accident Analysis & Prevention* **55**, 202–210 (2013).

[49] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. & Bruel, J.-M. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems (2009).

[50] Luo, Y. *et al.* A Driving Intention Prediction Method for Mixed Traffic Scenarios (2022).

[51] Fujiwara-Greve, T. in *Nash Equilibrium* (ed.Fujiwara-Greve, T.) *Non-Cooperative Game Theory* Monographs in Mathematical Economics, 23–55 (Springer Japan, Tokyo, 2015).

[52] Fudenberg, D. & Tirole, J. Noncooperative game theory for industrial organization: an introduction and overview. *Handbook of industrial Organization* **1**, 259–327 (1989).

[53] Lanctot, M. *et al.* A unified game-theoretic approach to multiagent reinforcement learning (2017). URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3323fe11e9595c09af38fe67567a9394-Paper.pdf.

[54] Huck, T. P., Ledermann, C. & Kröger, T. Simulation-based testing for early safety-validation of robot systems (2020).

[55] Koopman, P. & Wagner, M. Toward a framework for highly automated vehicle safety validation. Tech. Rep., SAE Technical Paper (2018).

[56] Abbas, H., O'Kelly, M., Rodionova, A. & Mangharam, R. Safe at any speed: A simulation-based test harness for autonomous vehicles (2019).

[57] Treiber, M., Hennecke, A. & Helbing, D. Congested traffic states in empirical observations and microscopic simulations. *Physical review E* **62**, 1805 (2000).

[58] Types of Unsignalized Intersections - Unsignalized Intersection Improvement Guide. https://toolkits.ite.org/uiig/types.aspx.

[59] National Highway Traffic Safety Administration. Query of fatality analysis reporting system (fars) web-based encyclopedia. https://www.nhtsa.gov/research-data/fatality-analysis-reporting-system-fars (2014). Accessed: 2014-12-04.

[60] Liberatore, S. Self-driving race car crashes into a wall from the starting line. https://www.dailymail.co.uk/sciencetech/article-8899021/Self-driving-race-car-crashes-straight-wall-starting-line-Roborace.html (2020).

[61] Kong, J., Pfeiffer, M., Schildbach, G. & Borrelli, F. Kinematic and dynamic vehicle models for autonomous driving control design (2015).

[62] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A. & Koltun, V. Carla: An open urban driving simulator (2017).

[63] Koenig, N. & Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator (2004).

[64] Brockman, G. *et al.* Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[65] Westhofen, L. *et al.* Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art. *Archives of Computational Methods in Engineering* **30**, 1–35 (2023).

[66] Minderhoud, M. M. & Bovy, P. H. Extended time-to-collision measures for road traffic safety assessment. *Accident Analysis & Prevention* **33**, 89–97 (2001).

[67] Saffarzadeh, M., Nadimi, N., Naseralavi, S. & Mamdoohi, A. R. A general formulation for time-to-collision safety indicator (2013).

[68] Choi, E.-H. Crash Factors in Intersection-Related Crashes: An On-Scene Perspective: (621942011-001) (2010).

[69] Langford, M. A., Zilberman, S. & Cheng, B. H.C.. Anunnaki: A modular framework for developing trusted artificial intelligence. *ACM Trans. Auton. Adapt. Syst.* (2024). URL https://doi-org.proxy2.cl.msu.edu/10.1145/3649453.

[70] Barry, K. Guide to Adaptive Cruise Control. https://www.consumerreports.org/cars/car-safety/guide-to-adaptive-cruise-control-a9154580873/ (2022).

[71] Company, F. M. Adaptive cruise control - setting the adaptive cruise control gap (2021). URL https://www.fordservicecontent.com/Ford_Content/vdirsnet/OwnerManual/Home/Content?variantid=7505&languageCode=en&countryCode=USA&Uid=G2047539&ProcUid=G2045342&userMarket=USA&

div=f&vFilteringEnabled=False&buildtype=web.

[72] Cao, Z. *et al.* Reinforcement learning based control of imitative policies for near-accident driving. *arXiv preprint arXiv:2007.00178* (2020).

[73] Li, N. *et al.* Game theoretic modeling of driver and vehicle interactions for verification and validation of autonomous vehicle control systems. *IEEE Transactions on control systems technology* **26**, 1782–1797 (2017).

[74] Gupta, P., Coleman, D. & Siegel, J. E. Towards safer self-driving through great pain (physically adversarial intelligent networks). *arXiv preprint arXiv:2003.10662* (2020).

[75] Zhou, W., Cao, Z., Deng, N., Jiang, K. & Yang, D. Identify, estimate and bound the uncertainty of reinforcement learning for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems* **24**, 7932–7942 (2023).

[76] Peters, H. *Game theory: A Multi-leveled approach* (Springer, 2015).

[77] Birchler, C., Khatiri, S., Bosshard, B., Gambi, A. & Panichella, S. Machine learning-based test selection for simulation-based testing of self-driving cars software. *Empirical Software Engineering* **28**, 71 (2023).

[78] Zheng, X. *et al.* Rapid generation of challenging simulation scenarios for autonomous vehicles based on adversarial test (2020).

[79] Zhong, Z., Kaiser, G. & Ray, B. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* (2022).

[80] Stocco, A., Nunes, P. J., d'Amorim, M. & Tonella, P. Thirdeye: Attention maps for safe autonomous driving systems (2022).

[81] Fremont, D. J. *et al.* Scenic: A language for scenario specification and data generation. *Machine Learning* **112**, 3805–3849 (2023).

[82] Liaskos, S., Khan, S. M., Golipour, R. & Mylopoulos, J. Towards goal-based generation of reinforcement learning domain simulations. (2022).

[83] Rudolph, S., Tomforde, S. & Hähner, J. On the detection of mutual influences and their consideration in reinforcement learning processes. *arXiv preprint arXiv:1905.04205* (2019).

[84] Jiang, H. *et al.* Multi-agent deep reinforcement learning with type-based hierarchical group communication. *Applied Intelligence* **51**, 5793–5808 (2021).

[85] Bruggner, D., Hegde, A., Acerbo, F. S., Gulati, D. & Son, T. D. Model in the loop testing and validation of embedded autonomous driving algorithms (2021).

[86] Leung, J., Shen, Z., Zeng, Z. & Miao, C. Goal modelling for deep reinforcement learning agents (2021).

[87] Cleland-Huang, J. *et al.* Human–machine Teaming with Small Unmanned Aerial Systems in a MAPE-K Environment. *ACM Transactions on Autonomous and Adaptive Systems* **19**, 1–35 (2024).

[88] Cámara, J., Moreno, G. & Garlan, D. Reasoning about Human Participation in Self-Adaptive Systems (2015).

[89] Arcaini, P., Riccobene, E. & Scandurra, P. Modeling and analyzing mape-k feedback loops for self-adaptation (2015).

[90] Gavidia-Calderon, C., Kordoni, A., Bennaceur, A., Levine, M. & Nuseibeh, B. The IDEA of Us: An Identity-Aware Architecture for Autonomous Systems. *ACM*

Transactions on Software Engineering and Methodology **33**, 1–38 (2024).

[91] Combemale, B. *et al.* A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems. *IEEE Software* **38**, 71–84 (2021).

[92] Franch, X. The i* framework: The way ahead (2012).

[93] Media, NHTSA. Distracted driving. https://www.nhtsa.gov/risky-driving/distracted-driving (2021).