# EvoDriver: Novelty-Search Driven Evolution of Behavioral Test Suites for Autonomous Vehicles

Anonymous Author(s)

## Abstract

In addition to environmental-based uncertainty conditions, self-adaptive Autonomous Vehicles (AVs) must be able to handle uncertainty posed by external human drivers sharing the operating environment. Existing search-based AV testing approaches have explored objective-oriented, top-down approaches to develop individual test cases to assess whether the system satisfies functional and safety properties with respect to specific operational contexts. However, these techniques lack search pressure to identify test cases that may be obscured in unintuitive regions of the search space (i.e., edge cases). This paper introduces a bottom-up, exploratory approach to automatic test case generation for AVs, where test cases are evolved organically (i.e., in a more open-ended fashion) to reveal uncertainty posed by diverse and unexpected maneuvers from an external vehicle in the operating environment. Specifically, novelty search-based genetic programming is used to evolve controllers for an external vehicle that interacts with the AV under study in order to reveal the most diverse behavioral responses from the AV under study. By leveraging diversity as a metric through *novelty search*, our approach automatically discovers behaviors for an external vehicle that result in increasingly different interactions with the AV under study. Our approach overcomes challenges typically associated with traditional search-based AV testing, such as developer bias and high development costs of manually tuning search objectives. We demonstrate the efficacy of our proof-of-concept framework in several traffic scenarios, discovering diverse test cases that enable us to reveal undesirable behaviors in the AV under study, including many edge cases that might otherwise not be detected.

## Keywords

online behavior-based testing, evolutionary computation, robustness, diversity, autonomous vehicles, novelty search, uncertainty, machine learning, cyber-physical systems

**ACM Reference Format:**
Anonymous Author(s). 2025. EvoDriver: Novelty-Search Driven Evolution of Behavioral Test Suites for Autonomous Vehicles. In *Proceedings of 21st International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '26).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 Introduction

In order to satisfy operational constraints and deliver acceptable behavior, Autonomous Vehicles (AVs) deployed in mixed-traffic environments (i.e., traffic involving different types of moving elements, including AVs and human-operated vehicles) must be able to handle different sources of uncertainty posed by other road users. AVs are self-adaptive [1–3], self-managing, cyber-physical systems [4, 5] that are safety-critical, where failures can lead to financial damage, bodily harm, and even fatal injuries to their occupants or nearby pedestrians and road users [6]. Existing AVs increasingly rely on black-box components, such as Machine Learning (ML) models, for key tasks such as perception, decision-making, and navigation. Black-box ML components introduce additional challenges, exhibiting poor statistical performance in some operating contexts, particularly those that involve humans and other uncertainty factors [7]. As such, a key software engineering challenge for self-adaptive cyber-physical systems such as AVs is how to comprehensively test their correctness over a large space of operating contexts, thereby discovering edge cases that might otherwise not be found using traditional testing approaches. By harnessing Evolutionary Computing (EC), this paper introduces a novelty search-based simulation testing framework for the automated assessment of AV robustness in the presence of uncertainty posed by diverse mixed-traffic interactions, including those that are unintuitive but plausible.

Safety for AVs deployed in a real-world setting is paramount as AVs must share the road with other external drivers that may exhibit erratic and unpredictable behaviors [8]. Previous research has largely focused on *offline testing* techniques that evaluate AVs by exposing them to historical data and assessing prediction errors in their outputs (e.g., failure to detect stop signs) [9, 10]. However, offline testing does not account for the closed-loop behavior of AVs,[1] which means they may fail to identify safety violations arising from complex interactions with the environment [9]. To avoid this problem, researchers have recently leveraged simulation-based approaches for *online testing* of closed-loop AV behavior in different operating contexts [9, 12, 13]. While significant advances have been made in the simulation of AV-operating environments, the modeling of external driver behavior remains a key challenge. One approach to behavioral modeling is rule-based models [14], where explicit control logic is used to define and govern the behavior of external road users. However, these approaches require significant development effort, may fail to capture complex interactions, and are susceptible to testing bias [15]. ML-based approaches (e.g., Reinforcement Learning (RL) [16, 17]) have been used to model driving behavior, but may lack interpretability and require significant computational resources for generating test cases [18]. Search-based testing techniques [19] look for test cases where certain developer-specified constraints or safety properties of the AV are violated, such as collisions or other safety violations of the AV under study [20–23]. However, by focusing on an explicit developer-specified objective (e.g., top-down searches for collisions), these approaches may only explore limited portions of the large and complex space of possible operating contexts [24], and may overlook unintuitive or corner cases of AV behavior that may be detrimental to safe operation of AVs.

[1]Closed-loop behavior refers to a feedback process where a system's outputs update the state of the system and the environment, thereby influencing future inputs [11].

This work introduces EvoDriver, an evolutionary search-based approach for automatically generating diverse behavior-based test cases for the AV under study. In this work, *diversity* is defined by the uniqueness of an observable behavior; that is, how much the behavior differs from those that have already been observed [25]. Existing techniques typically use a *top-down* approach to identify test cases based on an *a priori* specification of driving models/objectives and may suffer from developer bias and limit the scope of generated tests (e.g., maximize collisions). In contrast, EvoDriver is a *bottom-up* approach to AV testing where behaviors for an external vehicle are evolved based on the novelty of their impact on the AV's behavior (in comparison to what has been evolved thus far). The key insight is that by using *behavioral diversity* [25] to guide the evolutionary search, we are able to achieve unbiased and automated exploration of AV behavior in response to uncertainty posed by interactions with an external vehicle. As such, EvoDriver is a *complementary* approach to existing testing techniques, where we are able to uncover unintuitive/novel critical test cases that may otherwise not be found; critical test cases reveal unwanted behaviors in the AV, where criticality is with respect to safety properties [26, 27].

EvoDriver leverages EC [28] to automatically discover diverse test cases that result in a broad range of AV responses. This paper uses the term *ego vehicle* (i.e., Ego) to denote the AV under study and the term *non-ego vehicle* (i.e., NeV) to denote an external vehicle (human-operated or another deployed AV) with which the Ego interacts. In order to harness EC, EvoDriver uses Behavior Trees (BTs) [29] to represent and evolve behaviors for NeV(s) that pose a source of behavioral uncertainty to the Ego. BTs are mathematical models often used to specify the behavior of agents in simulation and gaming environments, where agent behaviors are encoded in a structured language [29]. We have developed a domain-specific BT representation language that enables EvoDriver to leverage Genetic Programming (GP) [30] to evolve NeV controllers(s). In our work, a traditional objective-based fitness function for the GP (e.g., safety, navigation, etc.) is replaced by a novelty metric that promotes diversity [25]. During the fitness evaluation, evolved BT-based NeV controllers that result in significantly different behaviors in the Ego are favored over those that lead to similar Ego behaviors. The result of applying our framework is a collection of distinct behaviors for the NeV that result in diverse interactions between the Ego under study and the evolved NeV in the operating context. By executing the Ego in the presence of those NeV(s), developers can discover unique failures that can inform concrete revisions to the Ego's requirements/software, including potential improvements to roadway infrastructure and traffic regulation.

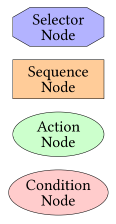
We demonstrate the ability of EvoDriver to discover diverse test cases in several use cases informed by real-world traffic data [31, 32]. Preliminary results show that EvoDriver can find a more diverse collection of critical test cases (i.e., near misses, collisions) when compared with other online search-based approaches [21, 33]. The remainder of the paper is organized as follows. Section 2 overviews enabling technologies used in this work. Section 3 describes the methodology of EvoDriver. Section 4 presents two use cases, highlighting the discovered uncertainty in different configurations. Section 5 discusses the key findings from our results. Section 6 overviews related work. Section 7 concludes our paper and previews future work.

## 2 Background

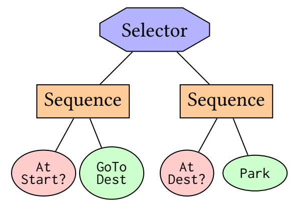
This section overviews enabling technologies used in this work. First, we overview the basic elements of BTs. Then we describe the evolutionary search techniques used in this work.

### 2.1 Behavior Trees

BTs are a popular technique used to formulate controllers for agents in simulation, robotics, and gaming settings [29]. Specifically, BTs provide control mechanisms to define the actions an agent may take through abstraction and hierarchical structure. BTs enable complex high-level behavior through control nodes and execution nodes, forming a tree-like structure. Figure 1(a) shows the types of nodes available in BTs. `Control` nodes are internal nodes that define how the tree should be traversed (i.e., *Selector* and *Sequence* nodes). A `Selector` node is an `OR` operator that attempts to execute any of the children, from left to right, until one succeeds or all fail. A `Sequence` node is an `AND` operator that executes the children in order, from left to right, until one fails or all succeed. `Action` nodes are leaf nodes of the tree, typically defining simple atomic actions or simple tasks. `Condition` nodes are leaf nodes of the tree that evaluate Boolean conditions. For example, `Condition` nodes may be used to determine whether the remaining children of a `Sequence` node should be executed. Finally, edges in a BT denote data flow and dependencies between nodes. Figure 1(b) shows an illustrative BT for a vehicle that selects between navigating to a destination or parking, depending on its current location.



**(a) BT legend.**



**(b) Sample BT.**

**Figure 1: Overview of BT node types and example BT.**

### 2.2 Evolutionary Computation

EC is a field of meta-heuristic optimization algorithms inspired by Darwinian evolution, where populations of solutions are iteratively evolved to improve their fitness [34]. In EC, individuals are encoded in structured formats (e.g., bitstrings, vectors, computation trees) known as *genotypes*. The observable behavior or characteristic corresponding to an individual's genotype is known as the *phenotype*. A *fitness function* measures optimality of a solution with respect to the target problem domain. By iteratively applying a set of evolutionary operators (e.g., mutation, crossover, and selection) to a randomly initialized population, EC discovers solutions with increasingly optimal fitness values. This work harnesses two subfields of EC to automatically discover diverse test cases for AV testing: GP [30] and novelty search [25].

*Genetic Programming.* GP is a class of evolutionary algorithms used to evolve computer programs [30]. In GP, genotypes encode program syntax trees comprising the hierarchical organization of primitive nodes (e.g., functions) and terminal nodes (e.g., constants or variables). The phenotypes are defined as the algorithm implemented by an individual's genotype. The *behavior space* is defined as the output produced by an individual's program when executed against a given set of input(s). While GP shares many similarities with genetic algorithms (GAs) [35], GPs typically evolve solution solvers (e.g., the vehicle controller that exhibits some desired properties). In contrast, GAs evolve solutions directly (e.g., a specific vehicle trajectory for a given scenario).

*Novelty Search.* Novelty search is a type of evolutionary algorithm that abandons the notion of objective fitness in favor of optimizing population diversity [25]. Novelty search is best suited for challenging problem domains without a clear strategy/objective for population improvement (i.e., deceptive landscapes [25]). The premise is that novelty search is better able to avoid getting stuck in local optima and instead capture a range of interesting (and unintuitive) solutions across the search space. Novelty search follows the same basic evolutionary process as traditional evolutionary algorithms, but replaces the fitness criteria with a *novelty metric*, rewarding individuals with high average distance to their nearest neighbors with better scores. In GP-based novelty search, the novelty metric may be formulated in terms of a difference in program structure or executable behavior.

## 3 Methodology

EvoDriver automatically discovers test cases that result in diverse behavior for an AV operating in a mixed-traffic scenario. Figure 2(a) shows the high-level Data Flow Diagram (DFD) for EvoDriver, where circles denote process bubbles, arrows denote information and data flow between processes, and parallel lines denote persistent data stores. Figure 2(b) provides details of the novelty computation process in **Step 3** (green bubble).

EvoDriver takes as input a specific traffic scenario, the black-box Ego controller, and a primitive set of driving actions (e.g., accelerating, changing lanes, etc.). For each traffic scenario, EvoDriver uses EC to generate a diverse collection of *test cases* in the form of *unique NeV BT controllers* to interact with the Ego controller to uncover unexpected and unwanted Ego behavior. The discovered test cases (i.e., NeV BT controllers) and corresponding test results (i.e., Ego behaviors when interacting with the NeV) support the assessment of Ego robustness and may provide developers with insights that can inform potential revisions of Ego requirements, applications of robustification techniques [36–38], or changes to road infrastructure and policies.

### 3.1 Testing Setup

This section overviews the setup and details of the inputs for the EvoDriver framework. The input to EvoDriver is a specific traffic scenario, the black-box Ego controller, and a primitive set of driving actions. The input *traffic scenario* is a concrete operating environment specification that includes a description of the static elements of the traffic environment and the participating vehicles (e.g., Ego and NeV) that operate in the environment. The static traffic environment is described in terms of a specification language (i.e., OpenDrive [39]) that includes road geometry, traffic regulations (e.g., speed limits, direction of travel), and semantic relationships between lanes (i.e., junctions). The participating vehicle descriptions include each vehicle's physical attributes (e.g., width, length, mass) and initial states (i.e., position, velocity, heading). Second, the input *Ego controller* is the software under study and can be any black-box executable controller (e.g., rule-based, RL-trained, end-to-end systems, etc.) that maps observations (e.g., environment states) to actions (e.g., throttle/steering values). Finally, the input *driving primitives* are a set of atomic driving actions and conditions (e.g., accelerating, turning, checking lane availability, etc.) that will be used to generate an NeV controller.

### 3.2 Behavior Tree Representation of Agents

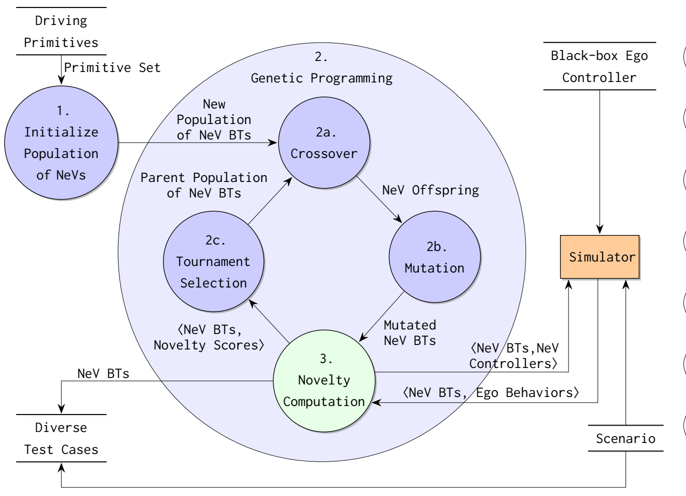
In order to discover previously unseen uncertainty, NeV controllers must be represented in a format that is amenable to GP-based evolution. To this end, EvoDriver uses BTs to encode NeV behaviors in a structured program representation (i.e., syntax tree). Specifically, BTs are codified as GP programs in our framework, which map sensor inputs to vehicle actions. EvoDriver's language is shown in Table 1. The GP program *primitives* are BT `Control` nodes (i.e., `Selector` nodes, `Sequence` nodes). The GP program *terminals* are BT execution nodes that include both `Condition` nodes and `Action` nodes. Each GP program terminal accepts zero or more parameters. For example, the ConstantVelocity `Action` may be parameterized by a target velocity ($v$) of 10 units/timestep and action duration ($d$) of 10 timesteps. If we allow any real-valued numbers to be used as parameters, then the effective exploration of the search space becomes computationally infeasible due to the complexity introduced by the combinatorial explosion [40]. To avoid this problem, we discretize a set of parameters for each GP terminal node, whose (empirically determined) values can be used by EvoDriver to instantiate the respective terminal elements. This insight allows EvoDriver to strategically explore the search space and discover unexpected and undesirable interactions.

Figure 3 shows a sample BT corresponding to an NeV agent that detects lane availability to perform a lane merge. The root of the BT is a `Selector` node that executes its children from left to right until one succeeds or all fail. In the example BT, each child (i.e., the left and right subtrees) is a `Sequence` node that executes its children from left to right, until one fails or all succeed. First, the root `Selector` node attempts to execute the left subtree. The left subtree specifies behavior that checks if the left lane is available. If so, then the NeV changes to the left lane and accelerates to a highway velocity of 60 units/timestep within 5 timesteps. If the left lane is not available, then the `Sequence` node fails and the `Selector` node moves to the right subtree. The right subtree specifies a behavior that reduces the vehicle's velocity to an on-ramp speed of 25 units/timestep over 8 timesteps and then comes to a stop.

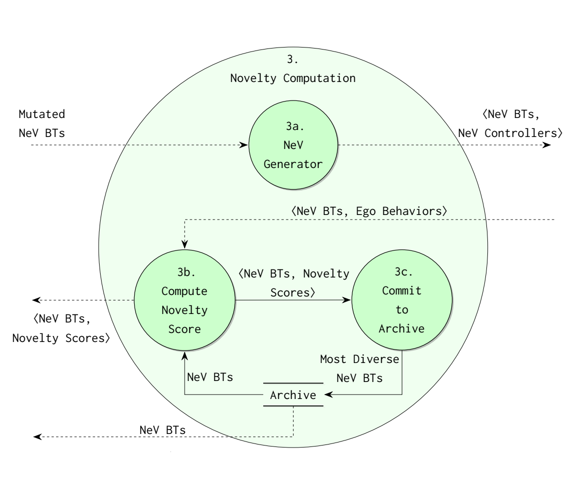
**Table 1: EvoDriver's GP Grammar**

| Name | Description |
|---|---|
| **Controls (GP Primitives)** | |
| `Selector(children)` | Execute children nodes until one succeeds. |
| `Sequence(children)` | Execute all children nodes in order. |
| **Actions (GP Terminals)** | |
| `ConstantVelocity(v, d)` | Drive at velocity $v$ for duration $d$. |
| `ChangeVelocity(v, d)` | Reach velocity $v$ within duration $d$. |
| `Turn(r, d)` | Steer $r$ degrees for duration $d$. |
| `Stop()` | Decelerate to 0 m/s. |
| `ChangeLane(l)` | Change to lane $l$, if lane $l$ is available. |
| **Conditions (GP Terminals)** | |
| `LaneAvailable(l)` | Check if lane $l$ is available. |
| `VehicleGap(c)` | Check if distance to nearest vehicle $\leq c$. |

| **Parameters** | |
|---|---|
| Variable | Possible Values |
| Velocity `v` | $\{5k : k \in \mathbb{Z}, 0 \leq k \leq 10\}$ |
| Duration `d` | $\{2k : k \in \mathbb{Z}, 0 \leq k \leq 10\}$ |
| Angle `r` | $\{15k : k \in \mathbb{Z}, -12 \leq k \leq 12\}$ |
| Lane `l` | $\{\texttt{lane.id} : \texttt{lane} \in \text{Roadway}\}$ |
| Vehicle gap `c` | $\{2k : k \in \mathbb{Z}, 10 \leq k \leq 20\}$ |
| List of `children` | $\{\texttt{node} : \texttt{node} \in \text{Terminal Nodes}\}$ |



(a) Overview for the evolution steps in EvoDriver.



(b) Novelty fitness metric computation in EvoDriver.

**Figure 2: Data flow diagram describing the EvoDriver framework.**

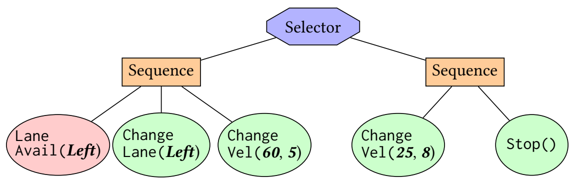
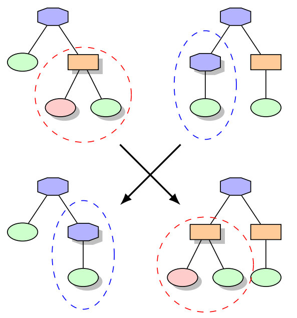


**Figure 3: Example BT for an NeV that attempts to merge and stops if the merge is not possible.**

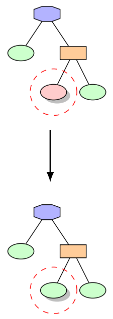
## 3.3 Discovering Diverse Behavior using GP

This section describes the EC process used in EvoDriver. Our framework largely follows a standard GP approach [30], where evolution continues until the maximum number of generations is reached. A key distinguishing difference with the standard GP approach is that EvoDriver maintains a separate archive of individuals that represents the most diverse individuals accumulated from all generations of the evolutionary search. Once the maximum number of generations has been reached, the resulting archive contains test cases that reveal diverse Ego behaviors.

**Step 1** initializes a population comprising a set of randomly generated NeV BTs. During each step of the GP, the current population of NeV BTs undergoes selection, crossover, and mutation. In **Step 2a**, parents selected for reproduction create offspring through a subtree crossover operator. Figure 4(a) shows an example subtree crossover operation, where a subtree from one parent is swapped with a randomly chosen subtree from the other parent (denoted by the red- and blue-dashed circles, respectively). Specifically, the crossover operation recombines the aggregate behaviors of its parents to obtain a novel NeV BT. Next, in **Step 2b**, new offspring have a predefined probability to undergo mutation. Figure 4(b) shows an example of the mutation operation, where a subtree of arbitrary size is replaced by a randomly-generated subtree (denoted by red-dashed circles). The mutation process introduces new NeV behaviors that potentially result in previously unseen responses from the Ego. The novelty score for the new population of NeV BTs is calculated in **Step 3** (see Section 3.4). The NeV BTs that yield the highest novelty scores are selected through tournament selection in **Step 2c** and then returned to **Step 2a** for crossover.



(a) Point crossover operation.



(b) Mutation operation.

**Figure 4: Overview of GP operations applied to BTs.**

## 3.4 Novelty Metric

This section describes how EvoDriver computes the novelty metric for each population of NeV BTs to guide the evolutionary search. Figure 2(b) shows details of **Step 3**, including substeps used to compute the novelty metric. In contrast to searching for behaviors that reflect a specific performance metric, EvoDriver rewards individuals for differing from previously seen behaviors in earlier generations [25]. This approach enables a more comprehensive exploration of the behavioral search space.

*Novelty Computation.* We formally define the novelty metric and describe how it is computed next. Let the current population $P$ comprise $N$ individual NeV BTs, each denoted as $p_i$, where $i$ ranges from 1 to $N$. Formally, $P$ is specified in Expression (1).

$$P := \{p_i : 1 \le i \le N\} \tag{1}$$

*Step 3a: NeV Generator.* In order to calculate the novelty score, an external simulator is initialized with a developer-specified traffic scenario and the black-box Ego controller. In **Step 3a**, EvoDriver transforms each NeV BT $p_i$ into an executable NeV controller and executes the simulator to observe Ego responses. EvoDriver tracks the Ego's state, $s^{(t)}$, during each timestep $t$ of the simulation. Expression (2) specifies the components of state $s^{(t)}$ that include the Ego's current position, velocity, and heading.

$$s^{(t)} = \langle \text{Ego}^{(t)}_{\text{pos}},\ \text{Ego}^{(t)}_{\text{velocity}},\ \text{Ego}^{(t)}_{\text{heading}} \rangle \tag{2}$$

For each NeV BT $p_i$, the simulator returns the corresponding Ego *behavior*, denoted as $\mathcal{B}_{p_i}$, in response to interactions with the NeV; Expression (3) specifies the simulator's output for a given NeV BT $p_i$. Specifically, the Ego behavior $\mathcal{B}_{p_i}$ corresponding to an NeV BT $p_i$ is the sequence of Ego states $s^{(t)}$, where timestep $t$ ranges from 1 to $T$, and $T$ is a maximum timestep parameter.

$$\mathcal{B}_{p_i} = \{s^{(t)} : 1 \le t \le T\}, \forall p_i \in P \tag{3}$$

*Step 3b: Compute Novelty Score.* EvoDriver uses the respective Ego's behavior $\mathcal{B}_{p_i}$ to compute the novelty scores of NeV BTs $p_i, \forall p_i \in P$. Expression (4) specifies the novelty metric. The novelty metric, $f(\cdot)$, is calculated using the average Euclidean distance between Ego behavior $\mathcal{B}_{p_i}$ and its $k$-nearest neighbors $B_{p_j}$, where $p_j \in P, \forall j \in [1, k], p_i \neq p_j$. We use $\mathcal{B}_P$ to denote the set of Ego behaviors corresponding to the current population $P$.

$$f(\mathcal{B}_{p_i}, \mathcal{B}_P, k) = \frac{1}{k} \sum_{j=1}^{k} \texttt{dist}(\mathcal{B}_{p_i}, \mathcal{B}_{p_j}) \tag{4}$$

*Step 3c: Commit to Archive.* NeV BTs $p_i$ with high novelty scores are committed to the archive, replacing those that have lower novelty scores. Finally, the population of NeV BTs and their respective novelty scores are returned to **Step 2c** to participate in the next generation of the evolutionary search process to iteratively discover the most diverse Ego behaviors. While the archive may initially include NeV BTs that have little impact on the Ego's behavior, NeV BTs that lead to similar Ego behaviors will be automatically discarded as the evolutionary search progresses. After multiple generations, the only way to improve population fitness is to find NeV BTs that result in novel Ego behaviors. Thus, by selecting for behavioral diversity, EvoDriver discovers NeV BTs that result in increasingly diverse Ego responses.

## 4 Demonstration Studies

This section describes our experimental setup and demonstrates several use cases to illustrate how our framework can be used to discover diverse and unintuitive test cases.

### 4.1 Experimental Setup

This section describes the experimental setup. For each use case, we apply EvoDriver to generate NeV behaviors for a specific mixed-traffic scenario. We compare EvoDriver with two alternative techniques for validation: Monte Carlo search [41] and Adversarial search [21]. Table 2 overviews the state-of-the-art GP parameters used in our validation studies for EvoDriver [25]. For Monte Carlo search, a set of 50 NeV BTs is randomly generated. For Adversarial search, we use the fitness objectives outlined in state-of-the-art approaches [21, 33, 42]. Expression (5) defines the fitness function used for Adversarial search, where `min_dist` is the minimum distance between the Ego and NeV for a given test case execution [42]. The 50 NeV BTs in the final archive from each approach form the basis for comparison.

$$f_{adv}(p_i) = \begin{cases} 1 & \text{if } \texttt{min\_dist} \le 0 \\ \frac{1}{\texttt{min\_dist}} & \text{else} \end{cases} \tag{5}$$

*Tooling.* We use several open-source tools and frameworks to assess the efficacy of EvoDriver to discover diverse test cases. We use the BARK simulator [43] to support simulation-based testing. The BARK simulator is lightweight and focuses on the observable behaviors of vehicles, rather than the high-fidelity graphics needed for perception tasks. The AV software provided by BARK is used as the Ego under study, where the AV software is an Intelligent Driver Model [43, 44] equipped with navigation, collision avoidance, and lane changing capabilities. We use the OpenDrive language for specifying the static scenario elements [39]. Experiments are conducted on a computer running Ubuntu 20 with an Intel i7 CPU and 32 GB of RAM.

**Table 2: GP hyperparameters used for experiments.**

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Population Size | 75 | Mutation Rate | 0.20 |
| Number of Generations | 100 | Crossover Rate | 0.85 |
| Archive Size | 50 | Tournament Size | 15 |
| k-Nearest | 3 | Max Arity | 3 |
| Novelty Threshold | 0.01 | Max Depth | 2 |

### 4.2 Evaluation Metrics

This section describes the metrics and definitions used for a *post hoc* evaluation of EvoDriver's generated test cases/results. The execution of a test case yields the observable behavior of the Ego when interacting with the NeV in a given scenario. In order to quantify Ego behavior, we use the AV criticality (i.e., safety) metrics outlined in existing literature [27, 45]. Specifically, we use the Time-To-Collision (TTC) metric [26, 46] to determine the criticality of a test case. The TTC value indicates the minimal time until two vehicles collide, assuming that they maintain their current trajectory. If no collision is predicted to occur, then the TTC value is set to infinity [26]. Expression (6) formally specifies the TTC metric, where $t$ and $\tilde{t}$ represent the current timestep and a future time offset, respectively; $p_{\text{Ego}}$ and $p_{\text{NeV}}$ represent the position of the Ego and the NeV, respectively. The function $d(\cdot)$ evaluates the Euclidean distance between the two input vehicle positions at timestep $t + \tilde{t}$.

$$\begin{aligned} TTC(\text{Ego}, \text{NeV}, t) = \min(\{\tilde{t} \ge 0 \mid \\ d(p_{\text{Ego}}(t+\tilde{t}), p_{\text{NeV}}(t+\tilde{t})) = 0\} \cup \{\infty\}). \end{aligned} \tag{6}$$

As part of *post hoc* analysis, we use domain knowledge from traffic safety research [26, 47, 48] to classify each test case result as one of the following *Ego behavior categories*:[2] Collision, Success, or Near Miss [50]. Expression (7) formally specifies the behavior categories, with respect to the minimum TTC observed over all timesteps $t$ for a given test case execution (i.e., $\text{TTC}_{\min}$). The Collision category describes test cases where the Ego experiences a collision. The Near Miss category describes test cases where the Ego is exposed to a dangerous event but does not collide. Specifically, a Near Miss scenario is defined as one where the TTC metric is below a given threshold $\tau$, where $\tau$ is selected based on values used in existing traffic safety research [26, 47, 48]. The Success category describes test cases where the Ego reaches its destination. Finally, we consider tests that result in the Collision category or the Near Miss category as *critical* test cases.

$$Cat(\text{TTC}_{\min}, \tau) = \begin{cases} \text{Collision} & \text{TTC}_{\min} = 0 \\ \text{Near Miss} & 0 < \text{TTC}_{\min} \le \tau \\ \text{Success} & \text{else} \end{cases} \tag{7}$$

### 4.3 Motivation for Diversity

Previous research in ML and software engineering has highlighted the importance of *diversity*, including the *benefits of balanced representations of different classes/categories of inputs*, for

[2]We use the phrase *Ego behavior categories* to describe the categories of traffic outcomes in a given scenario [49].

training/improving self-adaptive systems to handle previously unseen inputs [51–55]. A high diversity of test cases across and within behavior categories complements and supplements existing techniques that may exhibit uneven representation with respect to the categories of external vehicle behaviors [49]. As such, our experiments address the following research questions. **RQ1** explores if EvoDriver can discover diverse test cases that lead to different Ego behavior categories. Then for each Ego behavior category, **RQ2** explores the diversity of the test cases within a specific category.

**Research Questions**

**RQ1:** Can EvoDriver discover test cases that exhibit greater *inter-category* diversity than alternative approaches?
- $H_0\,(\mu_{\text{diff}}=0)$: There is no difference in inter-category diversity.
- $H_1\,(\mu_{\text{diff}}>0)$: There is a difference in inter-category diversity.

**RQ2:** Can EvoDriver discover test cases that exhibit greater *intra-category* diversity than alternative approaches?
- $H_0\,(\mu_{\text{diff}}=0)$: There is no difference in intra-category diversity.
- $H_1\,(\mu_{\text{diff}}>0)$: There is a difference in intra-category diversity.

## 4.4 Use Case: Merge Lane

Our first use case explores a merge scenario where a two-lane road reduces to a single lane. The U.S. National Highway Traffic Safety Administration (NHTSA) identifies merging as a frequent cause of traffic accidents [31, 56, 57]. Figure 5 provides an overview of the Merge Lane use case, where the orange NeV attempts to merge into the blue Ego's lane.

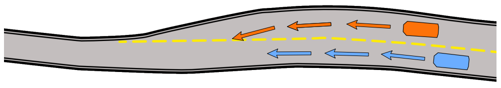


**Figure 5: Overview of the Merge Lane use case. The orange NeV must merge into the blue Ego's lane.**

*RQ1 - Inter-category diversity.* To address our first research question, we explore whether EvoDriver is able to discover a suite of test cases that are better distributed across the three Ego behavior categories (i.e., Collision, Success, and Near Miss) than what Monte Carlo search and Adversarial search can generate. Figure 6 shows the distribution of 50 generated test cases for each of the three approaches. The error bars show the standard deviation for each measured result across 10 trials. This result shows that EvoDriver is able to discover a more diverse (and evenly-distributed) test suite when compared to the two alternative approaches. The Adversarial search discovered only a single category of test cases, where the Ego failed to reach the destination due to a collision. The Monte Carlo technique only generated critical test cases approximately 38.8% of the time. In contrast, 67.6% of EvoDriver-generated test cases were critical, meaning that they led to an undesired Ego behavior due to the interaction with the NeV. Compared to the Adversarial search, EvoDriver discovered more evenly-distributed critical test cases that resulted in collisions or near misses for the Ego. Finally, Table 3 shows the mean, standard deviation, and p-value measured using the independent t-test for the diversity of test suites generated by each approach. The table shows that EvoDriver can consistently discover a test suite with higher novelty scores across ten experiments than the alternative approaches. Therefore, we found strong evidence (with $p < 0.01$) to reject our null hypothesis and support the alternate hypothesis for **RQ1**.

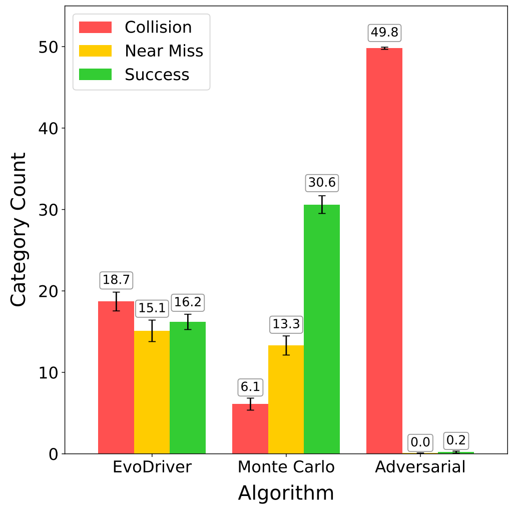


**Figure 6: Inter-category diversity measured for each of the approaches for the Merge Lane use case. EvoDriver discovered more evenly distributed test cases with respect to the behavior categories, including a greater number of Near Miss scenarios, compared to alternative approaches.**

**Table 3: Novelty score statistics for EvoDriver and alternative approaches for Merge Lane use case.**

| | **Novelty Score** (# test cases = 50) | | |
|---|---|---|---|
| Approach | Mean ($\mu$) | Std. Dev. ($\sigma$) | $p$-value ($\mu_{\text{diff}}$) |
| EvoDriver | **0.040** | 0.0026 | - |
| Adversarial | 0.011 | 0.0045 | **< 0.01** |
| Monte Carlo | 0.028 | 0.0008 | **< 0.01** |

*RQ2 - Intra-category diversity.* We also explore the diversity of test cases *within* each category of traffic outcomes. The measurement of *intra-category* diversity provides insights into the different types of NeV behaviors that can lead to the same category of Ego behavior. Figure 7 shows an analysis of the intra-category diversity of test cases. The number on the x-axis shows the average number of test cases for the given approach. For the Collision category, EvoDriver generated the most diverse set of test cases. Notably, EvoDriver discovered *more* diverse Ego failures with significantly *fewer* test cases (i.e., ≈ 62.5% fewer) when compared with the Adversarial search. For all categories, EvoDriver discovered more diverse intra-category test cases compared with alternative approaches. We found strong evidence (with $p < 0.01$) to reject our null hypothesis and support the alternate hypothesis for **RQ2**.

*Analyzing NeV behaviors.* Figure 8 shows the trajectories of each archive of test cases discovered by (a) EvoDriver, (b) Adversarial search, and (c) Monte Carlo search. Each trajectory shows the path taken by a single NeV in the archive. The colors of the trajectories indicate the category of outcome, where green represents Success, yellow represents Near Miss, and red represents Collision. Given the broad range of merging maneuvers discovered, these figures show that EvoDriver can discover the most diverse types of trajectories with respect to each Ego behavior category. Specifically, the trajectories observed from EvoDriver have greater coverage of the roadway and interactions for all three Ego behavior categories when compared with alternative approaches. In contrast, we observed a high degree of homogeneity in the trajectories generated by the Adversarial search, indicating that the search converged to similar behaviors (as indicated by the thick band of overlapping red trajectories) that led to collisions in the Ego. The Monte Carlo search led to random behaviors observed in the NeV, but most of these trajectories did not lead to critical cases for the Ego under study (i.e., number of green trajectories is majority of the cases found).

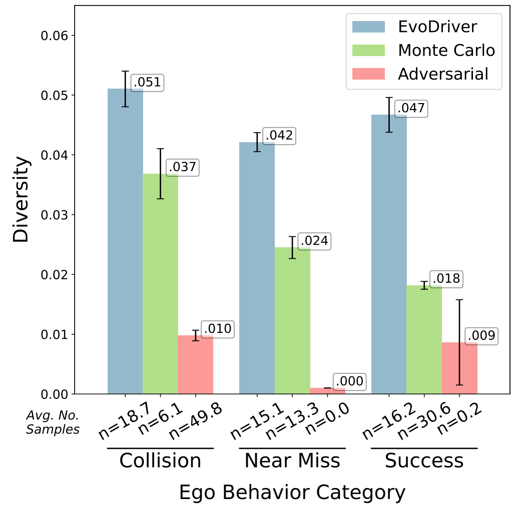
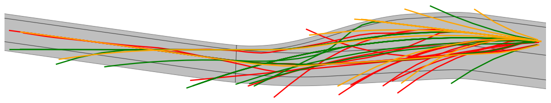


**Figure 7: Intra-category diversity measured for each of the search approaches for the Merge Lane use case.**

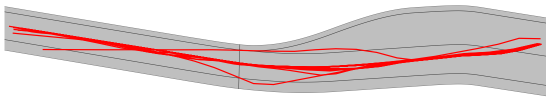
Figure 9 shows a test case of a Near Miss scenario discovered by EvoDriver with the corresponding evolved NeV BT. The trace plot in Figure 9(a) shows the blue Ego vehicle's near-miss interaction with an orange NeV that attempts to merge into the Ego's lane, where Figure 9(b) shows the BT corresponding to the orange NeV behavior visualized in the trace plot. A key capability of EvoDriver is the interpretability of NeV behaviors for a given test case, where the BT-based behavior representation can provide insights into the decision-making mechanisms of the NeV(s) when interacting with the Ego. First, the orange NeV initially attempts to merge into the left lane (see (A), Figures 9(a) and 9(b)). The maneuver fails as the left lane is occupied by the blue Ego. The root `Selector` node then attempts to execute its `Sequence` subtree (see (B), Figures 9(a) and 9(b)). The orange NeV accelerates to 40 units/timestep and then checks if the left lane is available again. The left lane is still not available, and thus the `Selector` node moves to the final branch. The final BT branch causes the orange NeV to turn 45 degrees left while maintaining its current velocity, leading to a Near Miss scenario where the blue Ego's collision avoidance feature results in a rapid deceleration in order to avoid a collision (see (C), Figures 9(a) and 9(b)). As such, the NeV behaviors discovered by EvoDriver may support developers in the robustification of the Ego, where we can use the BTs to guide our interpretation of NeV behaviors. For example, a robustification strategy may involve the introduction of a cautious mode that reduces the Ego's speed to allow aggressive merging vehicles to pass.

## 4.5 Use Case: Angled Intersection

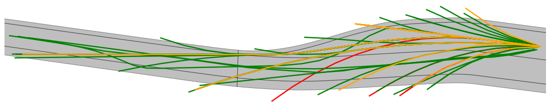
Our second use case explores an uncontrolled angled intersection where turning vehicles are required to make sharp turns. AVs are known to exhibit unexpected behavior in such intersections lacking traffic signals [32, 58, 59]. Figure 10 provides an overview of the Angled Intersection use case. The orange NeV attempts to cross the intersection while the blue Ego attempts to make a left turn.



**(a) Trajectory of NeVs discovered by EvoDriver.**

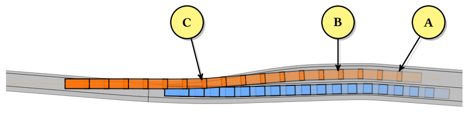


**(b) Trajectory of NeVs discovered by the Adversarial search.**

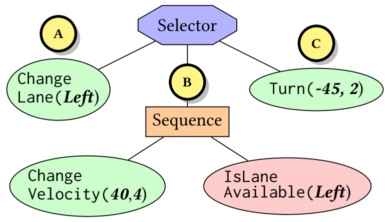


**(c) Trajectory of NeVs discovered by the Monte Carlo search.**

**Figure 8: Archive of NeVs discovered by each technique for the Merge Lane use case.**



**(a) Near Miss trajectory discovered by EvoDriver.**



**(b) BT evolved by EvoDriver.**

**Figure 9: Example of a Near Miss scenario and corresponding NeV BT discovered by EvoDriver.**

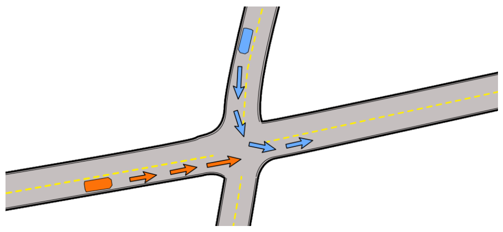


**Figure 10: Overview of the Angled Intersection use case.**

*RQ1 - Inter-category diversity.* We first compare the inter-category diversity between EvoDriver, Monte Carlo search, and Adversarial search. Notably, the Angled Intersection use case is more challenging to identify critical test cases due to the sparse behavior space, thus making the search problem well-suited for a diversity-driven approach. Figure 11 shows the count of test cases discovered for each category by each respective technique.

As before, EvoDriver discovered more evenly-distributed test cases when compared to the alternative techniques. Additionally, in comparison to the alternative approaches, EvoDriver discovered more test cases in the Near Miss category, where NeV behavior resulted in dangerous interactions with the Ego. Table 4 shows the average novelty scores of the final test suite for each technique. We found strong evidence (with $p < 0.01$) to reject our null hypothesis and support the alternate hypothesis for **RQ1**. As such, EvoDriver found the most diverse test suite.

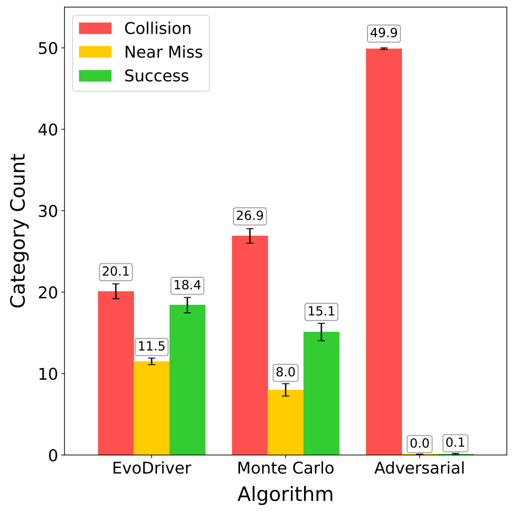


**Figure 11: Inter-category diversity measured for each of the search approaches for the Angled Intersection use case.**

**Table 4: Novelty score statistics for EvoDriver and alternative approaches for Angled Intersection use case.**

| | **Novelty Score** (# test cases = 50) | | |
|---|---|---|---|
| Approach | Mean ($\mu$) | Std. Dev. ($\sigma$) | $p$-value ($\mu_{\text{diff}}$) |
| EvoDriver | **0.059** | 0.002 | - |
| Adversarial | 0.002 | 0.001 | **< 0.01** |
| Monte Carlo | 0.047 | 0.003 | **< 0.01** |

*RQ2 - Intra-category diversity.* Next, we assess the intra-category diversity of the discovered NeV behaviors for the Angled Intersection use case. Figure 12 shows the diversity metric for the three categories discovered by each technique. As before, we found that EvoDriver discovered the most diverse failures compared with alternative approaches. Notably, EvoDriver discovered *more* diverse Ego failures with *fewer* test cases when compared with both Monte Carlo search and Adversarial search. For all categories, EvoDriver discovered more diverse intra-category test cases compared with alternative approaches. We found strong evidence (with $p < 0.01$) to reject our null hypothesis and support the alternate hypothesis for **RQ2**.

*Analyzing NeV behaviors.* Figure 13 shows the trajectories of each archive of test cases discovered by (a) EvoDriver, (b) Adversarial search, and (c) Monte Carlo search. Our results show that EvoDriver can discover NeV behaviors that result in the most diverse interactions with the Ego as both vehicles cross the intersection. Adversarial search converged to a set of NeV BTs that produced similar behavioral responses from the Ego and failed to cover all three behavior categories. Likewise, failures discovered by Monte Carlo search lacked diversity and generally had limited effect on Ego behavior. In contrast, EvoDriver discovered *diverse test cases* that covered *all* behavior categories.

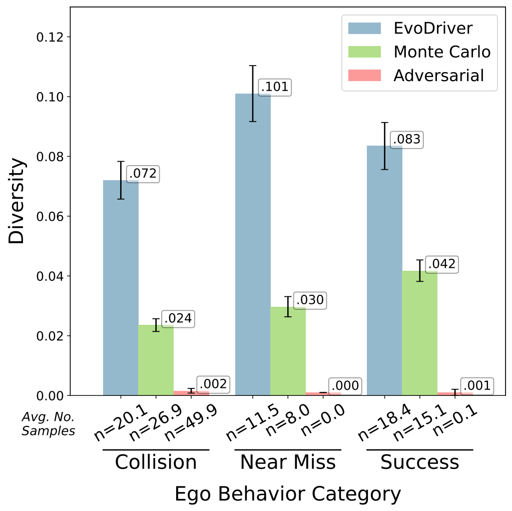
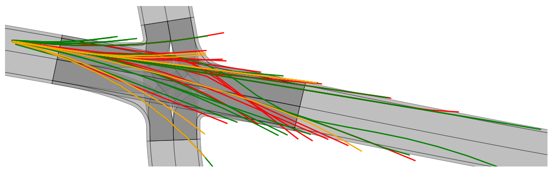
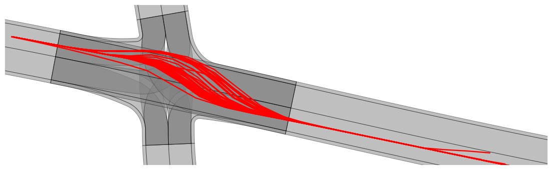


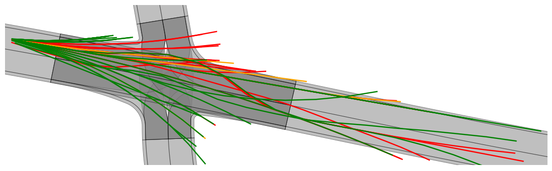
**Figure 12: Intra-category diversity measured for each of the search approaches for the Angled Intersection use case.**



**(a) Trajectory of NeVs discovered by EvoDriver.**



**(b) Trajectory of NeVs discovered by the Adversarial search.**



**(c) Trajectory of NeVs discovered by the Monte Carlo search.**

**Figure 13: Archive of NeVs discovered by each technique for the Angled Intersection use case.**

## 4.6 Threats to Validity

This paper harnessed GP with novelty search as an automated tool to discover test cases for an AV in a given operating context. The results of the experiments may vary with each evolutionary run, as EC exploits non-determinism to evolve solutions. In order to account for stochasticity and variance of our approach, each experiment demonstrated in our results is repeated 10 times. We used an average of each trial for the results, and found that the measured Coefficient of Variation for each trial was all less than 0.1. This indicates that repeated executions of experiments lead to consistent values in the measured metrics (i.e., TTC). As our experiments leveraged the BARK simulator for validation, we acknowledge that, as is commonly the case when using simulators, there may be possible deviations between agent behaviors observed in the simulator and reality (i.e., "reality gap" [60]). Finally, we used several use cases (i.e., different road architectures and objectives) to demonstrate EvoDriver's feasibility, flexibility, and context agnosticism.

## 5 Discussion

This section describes our analysis and findings from the EvoDriver use cases. Based on the results from both experiments, we found significant evidence to reject the null hypothesis for both **RQ1** and **RQ2**, and accept the alternative hypotheses that EvoDriver can find test cases that are more diverse, with respect to both inter- and intra-category diversity. Our *post hoc* analysis showed that EvoDriver can find test cases that lead to similar Ego responses as the Adversarial search, in addition to other unintuitive test cases that lead to more diverse near-miss scenarios. We next discuss the benefits of a bottom-up, open-ended approach to test case discovery and how the discovered test cases can be used by developers to improve AV robustness.

### 5.1 Bottom-up search for test cases

A key insight of our work is that by using a bottom-up search, we are more likely to discover unintuitive but plausible test cases than using a traditional top-down search approach. In top-down approaches, the observed Ego failures are often generated "by design" with respect to specific functional or non-functional objectives, thereby potentially introducing search biases and limiting exploration to a single dimension of Ego behaviors. In contrast, a bottom-up approach can organically uncover failures by exploring a broad range of NeV behaviors in an open-ended fashion. This strategic approach to test case generation reduces biases that would otherwise be introduced during objective specification in traditional top-down approaches. In addition, in the context of testing, the use of novelty search with its diverse archive promotes the discovery of test cases that 1) lead to respective Ego responses that are acceptable, dangerous, and adverse in a single instance of the experiment, 2) avoid deceptive landscapes to discover unique and unintuitive interactions, and 3) identify unexpected Ego behaviors in response to the NeV actions. Importantly, our search approach did not require us to specify in advance any search criteria about the Ego behavior categories, and thus the evolutionary process was not influenced by *a priori* information. Rather, the discovered Ego behaviors were categorized *post hoc* according to criteria used by state-of-the-art approaches in this field of study [27, 45]. EvoDriver was able to organically discover a set of different tests for each behavior category, where each behavior category-specific test suite showed greater diversity than alternative approaches. Additionally, EvoDriver may generate NeV behaviors that result in unpreventable collisions [61]. However, as NeV behaviors comprise high-level driving actions that follow realistic driving physics, AVs may encounter these unpreventable collisions when deployed. As such, the discovered NeV behaviors may inform AV developers of potential mitigation strategies to lessen/dampen the negative effects, even for unpreventable collisions [62].

Finally, a key advantage of EvoDriver when compared to existing AI-based testing frameworks [16, 63] is the *interpretability* of the generated NeV behaviors corresponding to each test case. Specifically, EvoDriver-generated BT-based controllers enable developers to retrospectively analyze the discovered NeV behaviors to gain insight into the explicit external vehicle decision-making logic that revealed undesirable Ego behaviors, thereby addressing a common limitation associated with existing AI-based testing frameworks. For example, in the Angled intersection use case, we demonstrated how a developer may analyze a discovered NeV BT to gain insights into specific NeV actions that led to undesirable interactions and responses from the Ego.

### 5.2 Towards AV Robustification

This section provides insight into how developers may use the output of EvoDriver to improve the robustness of AV software. EvoDriver's discovered diverse test cases expose the AV to a broad range of environmental contexts before deployment, thereby establishing/improving confidence in system safety; test results can then potentially be used to identify/mitigate potentially undesirable (latent) behaviors in response to uncertainty [49]. The key advantage of EvoDriver over Monte Carlo and Adversarial search is the ability to discover test cases that are diverse *across* and *within* behavior categories, including diverse examples for behavior categories that are challenging to explicitly uncover (i.e., near misses) [50]. As we move increasingly towards removing the human drivers from AVs (i.e., Society of Automotive Engineers (SAE) level 4 and 5 autonomy [64]), it becomes critically important to train the AVs for these edge cases because there will not be a human safety driver to provide a safeguard via manual override. For example, consider the test cases generated by EvoDriver for the Merge Lane use case, shown in Figure 8(a). The red and yellow trajectories represent the discovered uncertainty; that is, NeV behaviors that result in undesirable Ego responses. These diverse test cases discovered by EvoDriver can be used in different ways to robustify the Ego. First, these unintuitive but plausible NeV behaviors discovered by EvoDriver can be used for retraining learning-based Ego(s) (e.g., those trained with RL) in the presence of uncertainty, a robustification technique demonstrated by previous state-of-the-art approaches [16, 36, 49]. Second, developers can also use the EvoDriver-discovered uncertainty to inform revisions and/or additions of new requirements or operating constraints for the Ego to robustify against the discovered uncertainty. Finally, we can leverage EvoDriver-discovered diverse test cases covering distinct Ego behavior categories to train behavior oracles [49] to support run-time decision-making to detect dangerous external vehicle behaviors and trigger automated self-reconfiguration, such as switching to a fail-safe mode.

## 6 Related Work

This section overviews related work, including AV testing, novelty search, and other search-based AV testing techniques.

Recent research efforts have addressed the assurance of AVs. McDuff *et al.* [65] proposed CausalCity, a simulator framework that focuses on the explainability of ML-based AV models. However, they did not assess AV robustness against uncertainty. Fremont *et al.* [66] introduced Scenic and demonstrated a formal method approach to scenario-based test generation for AVs. Chan *et al.* [16] proposed SafeDriveRL, demonstrating that RL and non-cooperative game theory can be synergistically combined to discover AV test cases. However, their approach focused on discovering behaviors associated with a given human driving pattern, and thus did not consider the diversity of human driving patterns. Zheng *et al.* [67] proposed an optimization-based adversarial testing approach to generate testing scenarios for AVs, but did not explicitly address diversity. Zhong *et al.* [68] proposed a fuzz-based technique to generate scenarios for AV testing using a neural network. Jodat *et al.* [69] proposed using GP to evolve a test oracle for supporting simulation-based AV testing. Specifically, their test oracle infers whether a test case will pass/fail or require explicit execution in simulation to observe the outcome. However, existing approaches did not assess the robustness of AV against diverse and unexpected maneuvers of external human-operated vehicles.

Researchers have also explored various approaches using GPs to evolve agent controllers. Estgren and Jansson [70] demonstrated that GP can be used to evolve BTs, discovering agent controllers using high-level actions, leading to faster convergence. Smith *et al.* [71] showed that phenotypic novelty and objective fitness can be used in combination to produce controllers that adopt multiple strategies. However, their work evolved different controllers to solve a specific task, rather than controllers whose behaviors serve as test cases for a separate system under study. Iovino *et al.* [72] showed that GP can be used to learn the structure of a BT to solve robotic tasks. Montague *et al.* [73] proposed a hierarchical approach to evolve BTs using GPs, but their work did not concern diversity nor test case generation. These existing GP for BTs works did not address the testing of AVs and their interactions with human agents. They also generally use objective-based approaches. Lehman and Stanley [74, 75] first proposed the use of novelty search, where GPs are used to evolve diverse solutions. Naredo [76] explored a similar approach to evolve solutions for traditional ML tasks such as binary classification or regression analysis. Gomes *et al.* [77], Martinez *et al.* [78], and Velez *et al.* [79] showed how GP and novelty search can be used to evolve diverse agent controllers that solve particular tasks rather than evolving test cases for a system under study.

A number of researchers have explored search-based approaches to assess AV robustness. Gambi *et al.* [20] proposed a search-based framework used to generate road configurations to test AVs. However, their work concerns only the underlying infrastructure of the traffic scenario, and does not explore agent behaviors. Humeniuk *et al.* [80] propose a search-based approach to generate challenging static operating environments for autonomous systems by using RL to seed the initial EC population. Xie *et al.* [15] proposed a backward target-tree search technique to identify diverse test cases. Betts and Petty [24], Zhou *et al.* [81], and Tian *et al.* [33] presented genetic algorithm-based approaches to generate objective-based test cases for AVs. Huang and Nitschke [82] explored AV cooperation using novelty search, but did not study interactions that can lead to poor AV performance. Langford *et al.* [36] used novelty search to discover operational scenarios that lead to the most diverse system behavior, but did not consider external agents as part of their evolutionary search. Langford and Cheng [49] also used novelty search to identify environmental conditions (e.g., raindrops, lighting, etc.) that can lead to different categories of responses in the ML component. Chan and Cheng [83] applied a similar approach to generate diverse adversarial perturbations for ML components using novelty search. Li *et al.* [84] introduced SceGene, a genetic algorithm approach to discover test cases of dense traffic scenarios, but their work focused on generating a macroscopic traffic scene instead, in contrast to specific agent-level behavior. Schütt *et al.* [21] proposed a genetic algorithm approach to evolve BTs to discover scenario-based test cases. Yousefizadeh *et al.* [85] and Mandrioli *et al.* [86] proposed evolution-based test generation approaches, including co-evolution and GP, for assessing AV robustness, where parameterized scenarios are evolved to violate metamorphic testing-based relations for AV executions in critical scenarios. However, their top-down approach uses a developer-specified objective-based metric to evolve individuals, thereby introducing potential developer bias and limiting diversity. In contrast, our work focuses on discovering diverse test cases using a bottom-up approach, discovering combinations of atomic actions that can lead to increasingly different, unexpected, and unwanted behaviors in the AV under study.

Several previous approaches have explored coverage-based testing techniques to assess AV robustness [87]. Laurent *et al.* [88] proposed a coverage-based approach for testing parameterized rule-based AV decision-making components, where coverage was defined with respect to various safety- and comfort-based metrics. Majzik *et al.* [89] proposed situation coverage-based testing for AVs using graph-based scenario generation techniques, where coverage was defined with respect to domain-expert specified safety concepts. Tu *et al.* [90] proposed parameter coverage-based testing for AVs, where a Gaussian process was used to model the distribution of randomly generated traffic scenarios and then guide scenario sampling with respect to estimated safety criteria. These coverage-based approaches are best suited for exploring *a priori* developer-specified parameters and/or coverage criteria. Additionally, due to the large space of possible environmental contexts, coverage-based approaches commonly focus on a specific region of the search space [91] (e.g., low-level parameter enumeration such as the positioning of external vehicles), and may require a white-box testing setting to measure coverage criteria. In contrast, EvoDriver does not require *a priori* specification of search criteria, thus enabling us to automatically explore the search space to uncover unexpected/unintuitive AV behaviors in a black-box setting.

## 7 Conclusion

To deliver safe behavior, AVs deployed must safely handle uncertainty posed by humans in the operating context. This paper introduced EvoDriver, an AV testing framework for generating test cases to capture NeV behavior(s) that may lead to previously unseen or undesirable responses in the AV under study. We demonstrated EvoDriver in two different use cases, a merge lane and an angled intersection, two traffic scenarios that are known to be challenging for AVs [31, 32, 56–59]. We found that the test suites generated by EvoDriver were able to uncover diverse examples of unexpected Ego responses, including collisions and near misses. Importantly, EvoDriver discovered more diverse and unwanted Ego behaviors with fewer test cases when compared to alternative approaches. Bottom-up, diversity-driven evolution of test cases enabled us to overcome challenges associated with top-down approaches (e.g., search bias), where our generated interpretable test cases provide valuable insights to developers for revealing different ways the Ego may fail.

Future work includes the exploration of multi-agent scenarios (i.e., involving more than one external agent), additional traffic scenarios, and different road conditions (i.e., wet, icy, or muddy roads). We plan to explore how probabilistic programming techniques can be integrated with BTs to support formal specification of behavior spaces for rigorous verification of AV robustness in addition to run-time monitoring of external vehicle behavior. Finally, exploring techniques for failure trace analysis of the critical test cases may provide additional insights.

## References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM transactions on autonomous and adaptive systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[2] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software engineering for self-adaptive systems II: International seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised selected and invited papers*, pp. 214–238, Springer, 2013.

[3] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, (Zurich, Switzerland), pp. 99–108, IEEE, June 2012.

[4] H. Muccini, M. Sharaf, and D. Weyns, "Self-adaptation for cyber-physical systems: a systematic literature review," in *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems*, pp. 75–81, 2016.

[5] J. Al-Jaroodi, N. Mohamed, I. Jawhar, and S. Lazarova-Molnar, "Software engineering issues for cyber-physical systems," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 1–6, IEEE, 2016.

[6] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[7] M. Abdel-Aty and S. Ding, "A matched case-control analysis of autonomous vs human-driven vehicle accidents," *Nature Communications*, vol. 15, no. 1, p. 4931, 2024.

[8] P. Koopman and W. Widen, "Redefining Safety for Autonomous Vehicles," in *Computer Safety, Reliability, and Security* (A. Ceccarelli, M. Trapp, A. Bondavalli, and F. Bitsch, eds.), vol. 14988, pp. 300–314, Cham: Springer Nature Switzerland, 2024.

[9] F. U. Haq, D. Shin, S. Nejati, and L. Briand, "Can Offline Testing of Deep Neural Networks Replace Their Online Testing?," *Empirical Software Engineering*, vol. 26, p. 90, July 2021.

[10] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[11] S. Preuße, H.-C. Lapp, and H.-M. Hanisch, "Closed-loop system modeling, validation, and verification," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pp. 1–8, IEEE, 2012.

[12] W. Ding, C. Xu, M. Arief, H. Lin, B. Li, and D. Zhao, "A Survey on Safety-Critical Driving Scenario Generation—A Methodological Perspective," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, pp. 6971–6988, July 2023.

[13] H. Alghodhaifi and S. Lakshmanan, "Autonomous vehicle evaluation: A comprehensive survey on modeling and simulation approaches," *Ieee Access*, vol. 9, pp. 151531–151566, 2021.

[14] S. Masuda, H. Nakamura, and K. Kajitani, "Rule-based searching for collision test cases of autonomous vehicles simulation," *IET Intelligent Transport Systems*, vol. 12, no. 9, pp. 1088–1095, 2018.

[15] Y. Xie, Y. Zhang, C. Yin, H. Huang, and K. Dai, "Diversified Critical-Scenario-Search for Defect Detection of Autonomous Driving System," *IEEE Transactions on Intelligent Vehicles*, pp. 1–15, 2024.

[16] K. H. Chan, S. Zilberman, N. Polanco, J. E. Siegel, and B.H.C. Cheng, "SafeDriveRL: Combining non-cooperative game theory with reinforcement learning to explore and mitigate human-based uncertainty for autonomous vehicles," in *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 214–220, 2024.

[17] S. Feng, H. Sun, X. Yan, H. Zhu, Z. Zou, S. Shen, and H. X. Liu, "Dense reinforcement learning for safety validation of autonomous vehicles," *Nature*, vol. 615, no. 7953, pp. 620–627, 2023.

[18] C. Glanois, P. Weng, M. Zimmer, D. Li, T. Yang, J. Hao, and W. Liu, "A survey on interpretable reinforcement learning," *Machine Learning*, vol. 113, no. 8, pp. 5847–5890, 2024.

[19] P. McMinn, "Search-based software test data generation: a survey," *Software testing, Verification and reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[20] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, (Beijing China), pp. 318–328, ACM, July 2019.

[21] B. Schütt, M. Zhang, C. Steinhauser, and E. Sax, "Evolutionary Behavior Tree Generation for Dynamic Scenario Creation in Testing of Automated Driving Systems," in *2023 7th International Conference on System Reliability and Safety (ICSRS)*, pp. 322–330, Nov. 2023.

[22] K. Hao, W. Cui, Y. Luo, L. Xie, Y. Bai, J. Yang, S. Yan, Y. Pan, and Z. Yang, "Adversarial Safety-Critical Scenario Generation using Naturalistic Human Driving Priors," *IEEE Transactions on Intelligent Vehicles*, pp. 1–16, 2023.

[23] A. Wachi, "Failure-Scenario Maker for Rule-Based Agent using Multi-agent Adversarial Reinforcement Learning and its Application to Autonomous Driving," May 2019.

[24] K. M. Betts and M. D. Petty, "Automated search-based robustness testing for autonomous vehicle software," *Modelling and Simulation in Engineering*, vol. 2016, no. 1, p. 5309348, 2016.

[25] J. Lehman, K. O. Stanley, *et al.*, "Exploiting open-endedness to solve problems through the search for novelty.," in *ALIFE*, pp. 329–336, 2008.

[26] L. Westhofen, C. Neurohr, T. Koopmann, M. Butz, B. Schütt, F. Utesch, B. Neurohr, C. Gutenkunst, and E. Böde, "Criticality metrics for automated driving: A review and suitability analysis of the state of the art," *Archives of Computational Methods in Engineering*, vol. 30, no. 1, pp. 1–35, 2023.

[27] E. de Gelder and J.-P. Paardekooper, "Assessment of Automated Driving Systems using real-life scenarios," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 589–594, June 2017.

[28] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[29] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[30] W. B. Langdon and R. Poli, *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[31] B. Sen, J. D. Smith, W. G. Najm, *et al.*, "Analysis of lane change crashes," tech. rep., United States. Department of Transportation. National Highway Traffic Safety ..., 2003.

[32] X. Wang, D. Zhao, H. Peng, and D. J. LeBlanc, "Analysis of unprotected intersection left-turn conflicts based on naturalistic driving data," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 218–223, IEEE, 2017.

[33] H. Tian, Y. Jiang, G. Wu, J. Yan, J. Wei, W. Chen, S. Li, and D. Ye, "MOSAT: Finding safety violations of autonomous driving systems using multi-objective genetic algorithm," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (Singapore Singapore), pp. 94–106, ACM, Nov. 2022.

[34] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[35] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[36] M. A. Langford, G. A. Simon, P. K. McKinley, and B.H.C. Cheng, "Applying evolution and novelty search to enhance the resilience of autonomous systems," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 63–69, IEEE, 2019.

[37] Z. Wan, K. Swaminathan, P.-Y. Chen, N. Chandramoorthy, and A. Raychowdhury, "Analyzing and improving resilience and robustness of autonomous systems," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[38] M. Staron, *Automotive software architectures*. Springer, 2021.

[39] M. Dupuis, M. Strobl, and H. Grezlikowski, "Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks," in *Proc. of the Driving Simulation Conference Europe*, pp. 231–242, 2010.

[40] P. Schuster, "Taming combinatorial explosion," *Proceedings of the National Academy of Sciences*, vol. 97, no. 14, pp. 7678–7680, 2000.

[41] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd international conference on software engineering*, pp. 1–10, 2011.

[42] P. Zhang, L. Ming, T. Yuan, C. Qiu, Y. Li, X. Hui, Z. Zhang, and C. Huang, "Realistic Safety-critical Scenarios Search for Autonomous Driving System via Behavior Tree," May 2023.

[43] J. Bernhard, K. Esterle, P. Hart, and T. Kessler, "Bark: Open behavior benchmarking in multi-agent environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[44] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[45] H. Zhang and Y.-F. Li, "Integrated optimization of test case selection and sequencing for reliability testing of the mainboard of Internet backbone routers," *European Journal of Operational Research*, vol. 299, pp. 183–194, May 2022.

[46] M. M. Minderhoud and P. H. Bovy, "Extended time-to-collision measures for road traffic safety assessment," *Accident Analysis & Prevention*, vol. 33, no. 1, pp. 89–97, 2001.

[47] K. El-Basyouny and T. Sayed, "Safety performance functions using traffic conflicts," *Safety science*, vol. 51, no. 1, pp. 160–164, 2013.

[48] B. Huber, S. Herzog, C. Sippl, R. German, and A. Djanatliev, "Evaluation of virtual traffic situations for testing automated driving functions based on multidimensional criticality analysis," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, IEEE, 2020.

[49] M. A. Langford and B.H.C. Cheng, "'Know What You Know': Predicting behavior for learning-enabled systems when facing uncertainty," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 78–89, IEEE, 2021.

[50] T. Imaseki, F. Sugasawa, E. Kawakami, and H. Mouri, "Criticality metrics study for safety evaluation of merge driving scenarios, using near-miss video data," *SAE International Journal of Transportation Safety*, vol. 12, no. 09-12-01-0002, pp. 25–42, 2023.

[51] C. Liu, X. Yu, Y.-H. Tsai, M. Faraki, R. Moslemi, M. Chandraker, and Y. Fu, "Learning to learn across diverse data biases in deep face recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4072–4082, 2022.

[52] Y. Bengio, F. Bastien, A. Bergeron, N. Boulanger-Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, M. Côté, D. Erhan, J. Eustache, *et al.*, "Deep learners benefit more from out-of-distribution examples," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 164–172, JMLR Workshop and Conference Proceedings, 2011.

[53] M. A. Langford, S. Zilberman, and B.H.C. Cheng, "Anunnaki: a modular framework for developing trusted artificial intelligence," 2024.

[54] A. Rozsa, E. M. Rudd, and T. E. Boult, "Adversarial diversity and hard positive generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 25–32, 2016.

[55] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, pp. 1–42, 2013.

[56] H. Yang and K. Ozbay, "Estimation of traffic conflict risk for merging vehicles on highway merge section," *Transportation research record*, vol. 2236, no. 1, pp. 58–65, 2011.

[57] J.-S. Wang, *Lane change/merge crashes: problem size assessment and statistical description*. US Department of Transportation, National Highway Traffic Safety Administration, 1994.

[58] C.-Y. Chan, "Defining safety performance measures of driver-assistance systems for intersection left-turn conflicts," in *2006 IEEE Intelligent Vehicles Symposium*, pp. 25–30, IEEE, 2006.

[59] C. Sun, J. Leng, and B. Lu, "Interactive left-turning of autonomous vehicles at uncontrolled intersections," *IEEE Transactions on Automation Science and Engineering*, vol. 21, no. 1, pp. 204–214, 2022.

[60] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in artificial life: Third European conference on artificial life granada, Spain, june 4–6, 1995 proceedings 3*, pp. 704–720, Springer, 1995.

[61] P. G. Rees Jr, "Unavoidable accident–a misunderstood concept," *Ariz. L. Rev.*, vol. 5, p. 225, 1963.

[62] M. Parseh and F. Asplund, "New needs to consider during accident analysis: Implications of autonomous vehicles with collision reconfiguration systems," *Accident Analysis & Prevention*, vol. 173, p. 106704, 2022.

[63] M. Jiang, Y. Bai, A. Cornman, C. Davis, X. Huang, H. Jeon, S. Kulshrestha, J. Lambert, S. Li, X. Zhou, *et al.*, "Scenediffuser: Efficient and controllable driving simulation initialization and rollout," *Advances in Neural Information Processing Systems*, vol. 37, pp. 55729–55760, 2024.

[64] S. O.-R. A. V. S. Committee *et al.*, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems," *SAE Standard J*, vol. 3016, p. 1, 2014.

[65] D. McDuff, Y. Song, J. Lee, V. Vineet, S. Vemprala, N. A. Gyde, H. Salman, S. Ma, K. Sohn, and A. Kapoor, "Causalcity: Complex simulations with agency for causal discovery and reasoning," in *Conference on Causal Learning and Reasoning*, pp. 559–575, PMLR, 2022.

[66] D. J. Fremont, E. Kim, Y. V. Pant, S. A. Seshia, A. Acharya, X. Bruso, P. Wells, S. Lemke, Q. Lu, and S. Mehta, "Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World," July 2020.

[67] X. Zheng, H. Liang, B. Yu, B. Li, S. Wang, and Z. Chen, "Rapid generation of challenging simulation scenarios for autonomous vehicles based on adversarial test," 2020.

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

[69] B. A. Jodat, K. Gaaloul, M. Sabetzadeh, and S. Nejati, "Automated test oracles for flaky cyber-physical system simulators: Approach and evaluation," *arXiv preprint arXiv:2508.20902*, 2025.

[70] M. Estgren and E. S. V. Jansson, "Behaviour Tree Evolution by Genetic Programming." https://sciion.se/assets/papers/genetic-behaviour-trees.pdf.

[71] D. Smith, L. Tokarchuk, and C. Fernando, "Evolving Diverse Strategies Through Combined Phenotypic Novelty and Objective Function Search," in *Applications of Evolutionary Computation* (A. M. Mora and G. Squillero, eds.), (Cham), pp. 344–354, Springer International Publishing, 2015.

[72] M. Iovino, J. Styrud, P. Falco, and C. Smith, "Learning behavior trees with genetic programming in unpredictable environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4591–4597, IEEE, 2021.

[73] K. Montague, E. Hart, and B. Paechter, "A Hierarchical Approach to Evolving Behaviour-Trees for Swarm Control," in *Applications of Evolutionary Computation* (S. Smith, J. Correia, and C. Cintrano, eds.), (Cham), pp. 178–193, Springer Nature Switzerland, 2024.

[74] J. Lehman and K. O. Stanley, "Efficiently evolving programs through the search for novelty," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 837–844, 2010.

[75] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," *Genetic programming theory and practice IX*, pp. 37–56, 2011.

[76] E. Naredo, *Genetic programming based on novelty search*. PhD thesis, ITT, Instituto tecnologico de Tijuana, 2016.

[77] J. Gomes, P. Urbano, and A. L. Christensen, "Evolution of swarm robotics systems with novelty search," *Swarm Intelligence*, vol. 7, pp. 115–144, 2013.

[78] Y. Martínez, E. Naredo, L. Trujillo, and E. Galván-López, "Searching for novel regression functions," in *2013 IEEE congress on evolutionary computation*, pp. 16–23, IEEE, 2013.

[79] R. Velez and J. Clune, "Novelty search creates robots with general skills for exploration," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 737–744, 2014.

[80] D. Humeniuk, F. Khomh, and G. Antoniol, "Reinforcement learning informed evolutionary search for autonomous systems testing," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, pp. 1–45, 2024.

[81] R. Zhou, Y. Liu, K. Zhang, and O. Yang, "Genetic Algorithm-Based Challenging Scenarios Generation for Autonomous Vehicle Testing," *IEEE Journal of Radio Frequency Identification*, vol. 6, pp. 928–933, 2022.

[82] C.-L. Huang and G. Nitschke, "Evolutionary automation of coordinated autonomous vehicles," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–7, IEEE, 2020.

[83] K. H. Chan and B.H.C. Cheng, "Expound: A black-box approach for generating diversity-driven adversarial examples," in *International Symposium on Search Based Software Engineering*, pp. 19–34, Springer, 2023.

[84] A. Li, S. Chen, L. Sun, N. Zheng, M. Tomizuka, and W. Zhan, "SceGene: Bio-Inspired Traffic Scenario Generation for Autonomous Driving Testing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 14859–14874, Sept. 2022.

[85] H. Yousefizadeh, S. Gu, L. C. Briand, and A. Nasr, "Using cooperative co-evolutionary search to generate metamorphic test cases for autonomous driving systems," *IEEE Transactions on Software Engineering*, 2025.

[86] C. Mandrioli, S. Y. Shin, D. Bianculli, and L. Briand, "Testing cps with design assumptions-based metamorphic relations and genetic programming," *IEEE Transactions on Software Engineering*, 2025.

[87] Z. Tahir and R. Alexander, "Coverage based testing for v&v and safety assurance of self-driving autonomous vehicles: A systematic literature review," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 23–30, IEEE, 2020.

[88] T. Laurent, S. Klikovits, P. Arcaini, F. Ishikawa, and A. Ventresque, "Parameter coverage for testing of autonomous driving systems under uncertainty," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–31, 2023.

[89] I. Majzik, O. Semeráth, C. Hajdu, K. Marussy, Z. Szatmári, Z. Micskei, A. Vörös, A. A. Babikian, and D. Varró, "Towards system-level testing with coverage guarantees for autonomous vehicles," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 89–94, IEEE, 2019.

[90] J. Tu, S. Suo, C. Zhang, K. Wong, and R. Urtasun, "Towards scalable coverage-based testing of autonomous vehicles," in *Conference on Robot Learning*, pp. 2611–2623, PMLR, 2023.

[91] M. Boussaa, O. Barais, G. Sunye, and B. Baudry, "A novelty search-based test data generator for object-oriented programs," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1359–1360, 2015.