EvoDriver: Diversity-Driven Evolution of Behavioral Test Suites for Autonomous Vehicles

Anonymous Authors

***Abstract*—In addition to uncertainty due to environmental conditions, Autonomous Vehicles (AVs) must be able to handle uncertainty posed by humans sharing the operating context. 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 external agents in the operating context. Specifically, genetic programming is used to evolve controllers for external agents, where evolution is used to reveal the most diverse behaviors in the AV under study. By leveraging diversity as a metric through *novelty search*, our approach automatically discovers external vehicle behaviors 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 manual reward function tuning. 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 would otherwise not be detected.**

***Index Terms*—online behavior-based testing, evolutionary computation, robustness, diversity, autonomous vehicles, novelty search, uncertainty, machine learning**

I. Introduction

In order to satisfy operational constraints and deliver acceptable behavior, Autonomous Vehicles (AVs) deployed in mixed-traffic environments must be able to handle different sources of uncertainty posed by other road users (e.g., human drivers, pedestrians, bicyclists, etc.). However, 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 [1]. As such, a key challenge for AV development is how to comprehensively test their correctness over a large space of operating contexts, discovering edge cases that would otherwise not be found using traditional testing approaches. This paper introduces a diversity-driven search-based simulation testing framework for the automated assessment of AV robustness in the presence of uncertainty posed by a spectrum of diverse mixed-traffic interactions.

Safety for AVs deployed in a real-world setting is paramount [2]. Previous research has largely focused on *offline testing* techniques that evaluate AVs by exposing them to historical data and measuring errors in their outputs (e.g., failure to detect stop signs) [3], [4]. 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 [3]. To this end, researchers have recently leveraged simulation-based approaches for *online testing* of closed-loop AV behavior in different operating contexts [3], [6], [7]. 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 [8], where explicit control logic is used to define and govern the behavior of road users. However, these approaches require significant development effort, fail to capture complex interactions, and are susceptible to developer or tester bias [9]. ML-based approaches (e.g., Reinforcement Learning (RL) [10], [11]) have been used to model driving behavior, but may lack interpretability and require significant computational resources for generating test cases [12]. Search-based testing techniques [13] seek to identify test cases where certain developer-specified constraints or safety properties of the AV are violated. These existing approaches often focus on searching specifically for failure-inducing test cases, where failures can be collisions or safety violations by the AV under study [14], [15], [16], [17]. However, by focusing on an explicit developer-specified objective (e.g., top-down searches for failures), existing approaches may only be exploring limited operating contexts, thereby resulting in a narrow (and biased) range of discovered AV behaviors. As such, existing techniques have not been *designed* to discover unintuitive or corner cases of AV behavior that may be detrimental to the safe operation of AVs.

This work introduces EvoDriver, an evolutionary search-based approach for automatically generating diverse behavior-based test cases for the AV under study. Our approach discovers a number of driving behaviors exhibited from external vehicles that may pose uncertainty for the AV and lead to unexpected AV behaviors. The key insight is that using *diversity* [18] as the evolution objective is better suited to achieve unbiased and automated exploration of AV behavior in response to uncertainty posed by interactions with external vehicles. In contrast to existing *top-down* approaches where a fitness objective is specified and relevant test cases are identified (e.g., maximize collisions), EvoDriver is a *bottom-up* approach to AV testing where external vehicle behaviors

[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 [5].

are evolved based on the novelty of their impact on the AV's behavior. A key advantage of a bottom-up approach is removing the need for *a priori* specification of driving models/objectives that may suffer from developer bias and limit the scope of generated tests. Instead, EVODRIVER leverages the *behavioral* diversity of the AV under study as a selection metric to promote the discovery of novel test cases in previously unexplored regions of the search space that are difficult to find with traditional testing approaches. To this end, EVODRIVER is a *complementary* approach to existing testing techniques, which is able to uncover a comprehensive set of critical test cases[2] that would otherwise not be found.

EVODRIVER leverages Evolutionary Computation (EC) [22] 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 *non-ego vehicle* (i.e., NEV) to denote external vehicle(s) with which the EGO interacts. In order to harness EC, EVODRIVER uses Behavior Trees (BTs) [23] 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 [23]. We have developed a domain-specific BT representation language that enables EVODRIVER to leverage Genetic Programming (GP) [24] 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 [18]. During the fitness evaluation, evolved BT-based NEV controllers that result in significantly different behaviors in the AV are favored over those that lead to similar behaviors. The result of applying our framework is a collection of NEV(s) with distinct behaviors that result in diverse interactions between the EGO under study and the evolved NEVs 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 and/or software.

We demonstrate the ability of EVODRIVER to discover diverse test cases in several use cases. Preliminary results show that EVODRIVER can find a more diverse collection of critical test cases (i.e., near misses, collisions) when compared with other search-based approaches [15], [25]. The remainder of the paper is organized as follows. Section II overviews the background and enabling technologies used in this work. Section III describes the methodology of our proposed framework. Section IV demonstrates two use cases, where we highlight the discovered uncertainty in different configurations. Section V discusses the key findings from our results. Section VI overviews related work. Section VII considers our threats to validity. Section VIII concludes our paper and previews future work.

II. BACKGROUND

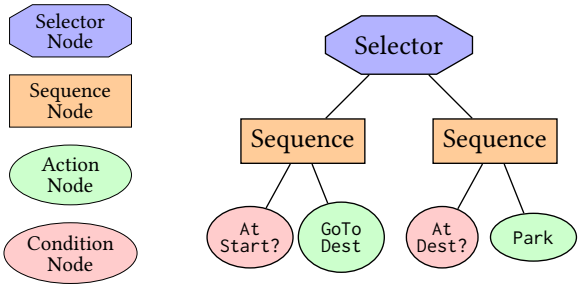
This section overviews the background material used in this work. First, we describe existing challenges that AVs face when deployed in a real-world setting. Second, we introduce BTs. Finally, we describe the evolutionary search techniques used in this work.

A. Challenges in Autonomous Driving

To prevent injuries and damage to the environment, AVs must deliver operationally correct and safe behavior when deployed in a real-world setting. As AVs and other humans share the road, AVs must operate correctly in the presence of other human road users, who may exhibit erratic and unpredictable behaviors. These behaviors serve as a source of uncertainty for deployed AVs. In order to ensure safe behavior, AVs must be rigorously tested for operational correctness to uncover and mitigate unsafe states before deployment. However, AV testing is difficult, as the space of possible operating contexts is large [26]. As such, testing techniques that can strategically navigate the complex search space to discover critical test cases are needed.

B. Behavior Trees

BTs are a popular technique used to formulate controllers for agents in simulation, robotics, and gaming settings [23]. 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 1a 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). Selector nodes attempt to execute any of the children until one succeeds or all fail (i.e., an OR operator). Sequence nodes execute the children in order until failure or all succeed (i.e., an AND operator). 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. Finally, edges in a BT denote statuses and dependencies. Figure 1b 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.

Fig. 1: Overview of BT node types and example BT.

C. 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 [27]. 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

[2]Critical test cases reveal unwanted behaviors in the AV, where criticality is defined with respect to some safety property [19], [20], [21].

fitness function measures the optimality of a given 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 generates solutions with increasingly optimal fitness values. This work harnesses two subfields of EC, GP [24] and novelty search [18], to automatically discover diverse test cases for AV testing.

a) Genetic Programming: GP is a class of evolutionary algorithms used to evolve computer programs [24]. In GP, genotypes encode program syntax trees comprising the hierarchal 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, GPs typically evolve solution solvers (e.g., the vehicle controller that exhibits some desired properties). In contrast, genetic algorithms evolve solutions directly (e.g., a specific vehicle trajectory for a given scenario).

b) Novelty Search: Novelty search is a type of evolutionary algorithm that abandons the notion of objective fitness in favor of optimizing population diversity [18]. Novelty search is best suited for challenging problem domains without a clear means of population improvement (i.e., deceptive landscapes [18]). The premise is that novelty search is better able to avoid local optima and capture a range of interesting (and unintuitive) solutions across the search space. Novelty search follows the same structure 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 the program's executable behavior.

III. Methodology

EvoDriver automatically discovers test cases that result in diverse behavior for an AV operating in a mixed-traffic scenario. Figure 2a 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 2b provides details of the novelty computation process in **Step 3** (green bubble). Next, we next describe each step of EvoDriver in turn.

A. Testing Setup

This section overviews the setup and input to the EvoDriver framework. First, a developer specifies a target traffic scenario. In this work, a *scenario* is defined by a static traffic environment and a set of dynamic agents (i.e., vehicles) that operate in the environment. The traffic environment is represented using a specification language (i.e., OpenDrive [28]) that includes road geometry, traffic regulations (e.g., speed limits, direction of travel), and semantic relationships between lanes (i.e., junctions).

The vehicles participating in the scenario are the dynamic agents whose behaviors and interactions pose a source of uncertainty for the Ego. The scenario's vehicle data describes each vehicle's physical attributes (e.g., width, length, mass) and initial states (i.e., position, velocity, heading). EvoDriver uses a *simulator* to assess the behavior of the Ego in a given scenario. The Ego is provided as input by a developer and can be any black-box model (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).

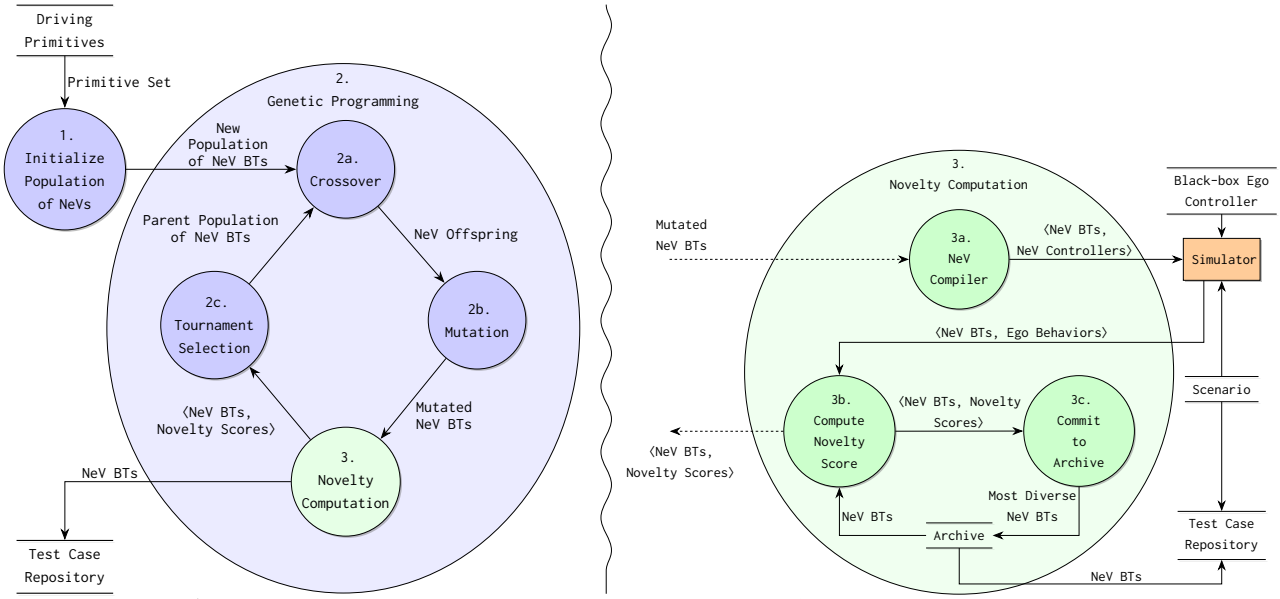
B. Behavior Tree Representation of Agents

In order to discover previously unseen uncertainty, NEV controllers must be represented in a format that facilitates 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 that map sensor inputs to vehicle actions. EvoDriver's language is shown in Table I. 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 conditions and actions. Each GP program terminal accepts zero or more parameters. For example, the ConstantVelocity action may be parametrized by a target velocity (v) of $10m/s$ and action duration (d) of $10s$. 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 [29]. To this end, we discretize a set of parameters for each GP terminal node, whose (empirically determined) values can be used by EvoDriver to instantiate the respective terminals. This insight allows EvoDriver to strategically explore the search space and discover undesirable interactions. Figure 3 shows a sample BT corresponding to an NEV agent that detects lane availability to perform a lane merge maneuver. Specifically, the left branch specifies behavior that checks if the left lane is available. If so, then the NEV changes to the left lane and executes a (slight) left turn with an angle of 15° for 4 timesteps. The right branch specifies a behavior that maintains a constant velocity of 30 units/timestep for 10 timesteps and then comes to a stop.

TABLE I: 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`	{`lane.id` : `lane` ∈ Roadway}
Vehicle gap `c`	$\{2k : k \in \mathbb{Z},\ 10 \leq k \leq 20\}$
List `children`	{`node` : `node` ∈ Terminal Nodes}



(a) DFD overview for the evolution steps in EvoDriver. (b) Novelty fitness metric computation in EvoDriver.

Fig. 2: Data flow diagram describing the EvoDriver framework.

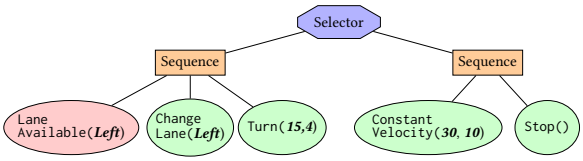


Fig. 3: Example BT for an NEV that attempts to merge and stops if the merge is not possible.

C. Discovering Diverse Behavior using GP

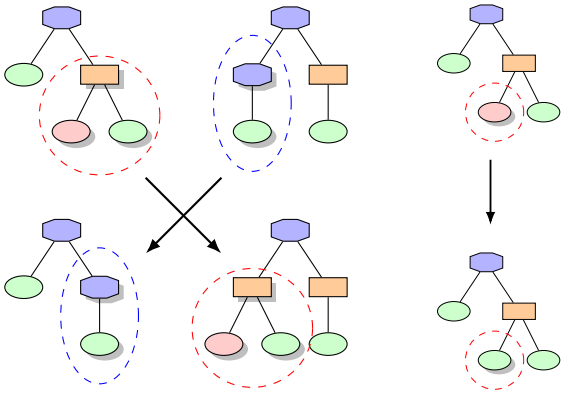
This section describes the evolutionary process used in EvoDriver. Our framework largely follows a standard GP approach, where evolution continues until the maximum number of generations is reached. EvoDriver maintains a separate archive of individuals that cumulates the most diverse individuals from any generation of the evolutionary search. Once the maximum number of generations has been reached, the resulting archive contains test cases that lead to diverse behaviors in the Ego. We next describe each step of the evolutionary search in turn.

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 4a 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). 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 4b 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 fitness score for the new population of NEV BTs is calculated in **Step 3** (see Section III-D).

The NEV BTs that yield the highest fitnesses (i.e., highest novelty scores) are selected through tournament selection in **Step 2c** and then returned to **Step 2a** for crossover.

D. Novelty Metric

This section describes how EvoDriver computes the novelty metric for the current population of NEV BTs to guide the evolutionary search. Figure 2b shows the details of **Step 3**, including the substeps used to compute the novelty metric. In contrast to searching for behaviors that reflect a specific performance metric, EvoDriver rewards individuals for diverging from previously seen behaviors in earlier generations [18]. This approach enables a more comprehensive exploration of the behavioral search space.



(a) Point crossover operation. (b) Mutation operation.

Fig. 4: Overview of GP operations applied to BTs.

a) 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.

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

b) Step 3a: NeV Compiler: In order to calculate the novelty score, a simulator is initialized with a developer-specified scenario and the black-box Ego controller. In

Step 3a, EVODRIVER compiles 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) formally captures 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 t ranges from 1 to T, and T is a maximum timestep parameter.

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

c) 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 current population P.

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

d) Step 3c: Commit to Archive: NEV BTs p_i with high novelty scores are committed to the archive, replacing those that have lower scores. Finally, the population of NEV BTs and their respective novelty scores are returned to **Step 2c**, allowing for evolutionary search 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 continues. 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.

IV. DEMONSTRATIONS

This section describes our experimental setup and demonstrates several use cases to validate our framework. We first describe the experiment settings, tooling, and evaluation metrics. Next we detail each use case in turn.

A. Experiment Settings

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 [30] and Adversarial search [15]. Table II overviews the state-of-the-art GP parameters used in our validation studies for EVODRIVER [18]. For Monte Carlo search, a set of 50 NEV BTs are randomly generated. For Adversarial search, we use the fitness objectives outlined in existing state-of-the-art approaches [15], [25], [31]. 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 [31]. The 50 most fit NEV BTs from each approach form the basis for comparison.

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

a) 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 [32] 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 [32], [33] equipped with navigation, collision avoidance, and lane changing capabilities. We use the OpenDrive language for specifying the static scenario elements [28]. Experiments are conducted on a computer running Ubuntu 20 with an Intel I6 CPU and 32 GB of RAM.

TABLE II: 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

B. Evaluation Metrics

This section describes the metrics and definitions used to evaluate EVODRIVER. A test case comprises a specific NEV BT, a corresponding traffic scenario, and the EGO under study. The execution of a test case yields the observable behavior of the EGO when interacting with the NEV in a specific scenario. In order to quantify EGO behavior, we use the AV criticality metrics outlined in existing literature [20], [21]. Specifically, we use the Time-To-Collision (TTC) metric [19], [34] to determine the criticality of a test case. The TTC value indicates the minimal time until two vehicles collide, assuming they maintain their current trajectory. If no collision is predicted to occur, then the TTC value is set to infinity [19]. Expression (6) formally specifies the TTC metric.

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

For a given test case, we classify it as one of the following EGO *behavior categories*:[3] COLLISION, SUCCESS, or NEAR MISS. 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., TTC_{min}). The COLLISION category describes test cases where the EGO experiences a collision. The NEAR MISS

[3]We use the phrase EGO *behavior categories* to describe the categories of traffic outcomes in a given scenario [35]

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 τ. 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.

$$Cat(\text{TTC}_{\text{min}}, \tau) = \begin{cases} \text{COLLISION} & \text{TTC}_{\text{min}} = 0 \\ \text{NEAR MISS} & 0 < \text{TTC}_{\text{min}} \leq \tau \\ \text{SUCCESS} & \text{else} \end{cases} \quad (7)$$

Our experiments address the following research question. **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 category-specific set of test cases.

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 the inter-category diversity.

$H_1(\mu_{\text{diff}} > 0)$: There is a difference in the 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 the intra-category diversity.

$H_1(\mu_{\text{diff}} > 0)$: There is a difference in the intra-category diversity.

C. Use Case: Merge Lane

Our first use case explores a merge scenario where a two-lane road reduces into a single lane. The U.S. National Highway Traffic Safety Administration (NHTSA) identifies merging as a frequent cause of traffic accidents [36], [37], [38]. Figure 5 provides an overview of the Merge Lane use case, where the orange NEV attempts to merge into the blue EGO's lane.

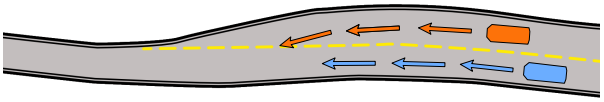


Fig. 5: Overview of the Merge Lane use case. The orange NEV must merge into the blue EGO's lane.

a) 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 Monte Carlo search and Adversarial search. Figure 6 shows the distribution of 50 generated test cases for all 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 (i.e., 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 impact on the EGO's behavior. 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 III shows the mean, standard deviation, and p-value (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**.

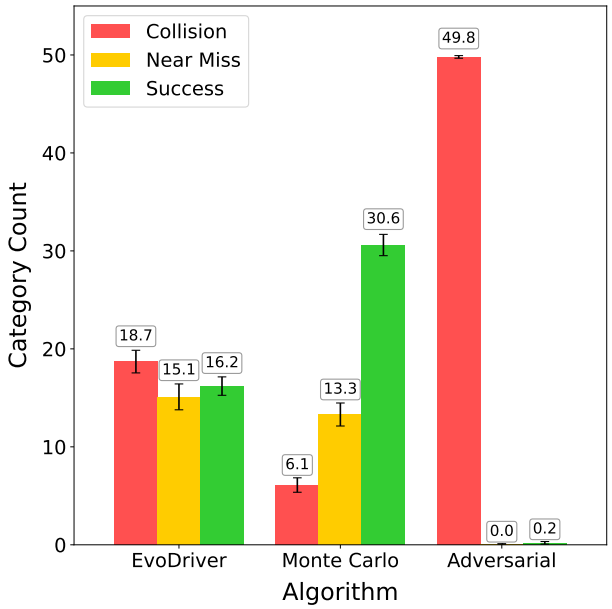


Fig. 6: The inter-category diversity measured for each of the search approaches for the Merge Lane use case.

TABLE III: Novelty score statistics for EVODRIVER and alternative approaches for Merge Lane use case.

Approach	Novelty Score		
	Mean (μ)	Std. Dev. (σ)	p-value (μ_{diff})
EVODRIVER	**0.040**	0.0026	-
Adversarial	0.011	0.0045	$< \mathbf{0.01}$
Monte Carlo	0.028	0.0008	$< \mathbf{0.01}$

b) 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., $\approx 62.5\%$ less) 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**.

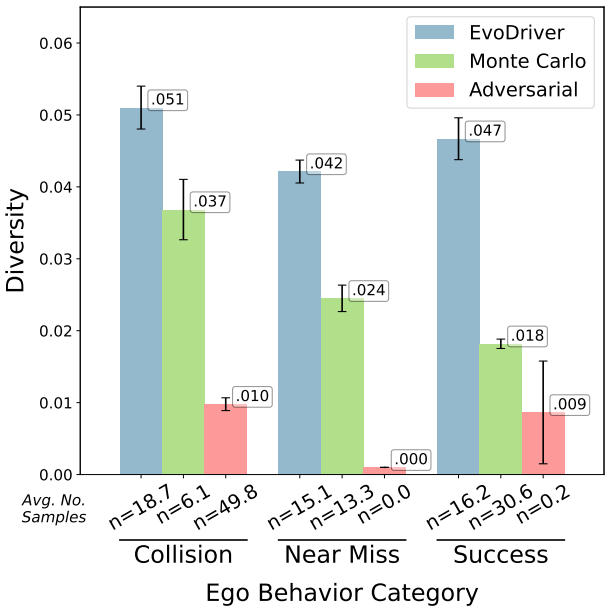
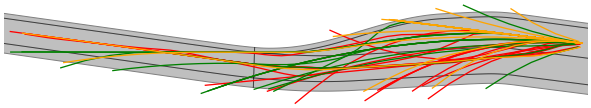


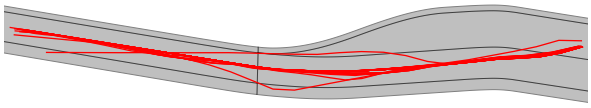
Fig. 7: The intra-category diversity measured for each of the search approaches for the Merge Lane use case.

c) 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 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., the number of green trajectories is the majority of the cases found).

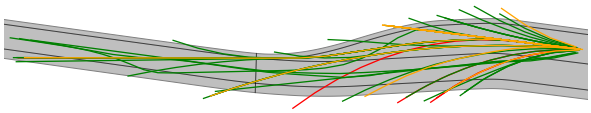
Finally, a key advantage of EvoDriver is the ability for developers to retrospectively analyze the discovered BTs to gain insight on the explicit actions that resulted in undesirable Ego behaviors (i.e., interpretability), thereby addressing a common limitation associated with existing testing frameworks [10]. To this end, Figure 9 shows an instance of a Near Miss scenario discovered by EvoDriver with the corresponding NEV BT that was evolved. The trace plot in Figure 9a shows the blue Ego vehicle's interaction with an orange NEV that attempts to merge into the Ego's lane. Specifically, EvoDriver discovered a test case where the orange NEV abruptly cuts off the blue Ego in order to merge. Figure 9b shows the BT corresponding to the NEV behavior visualized in the trace plot. First, the NEV initially attempts to merge into the left lane (see (A), Figure 9). The maneuver fails as the left-lane is occupied by the Ego. The root selector node then attempts to execute its sequence subtree (see (B), Figure 9). The NEV accelerates to 40 units/second 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 NEV to turn 45 degrees left while maintaining its current velocity, leading to a Near Miss scenario where the Ego must quickly decelerate in order to avoid a collision (see (C), Figure 9). Notably, the interpretability of NEV behaviors discovered by EvoDriver may support developers in the robustification of the Ego, for example, introducing a cautious mode that reduces the Ego's speed to allow aggressive merging vehicles to pass.



(a) Trajectory of NEVs discovered by EvoDriver.

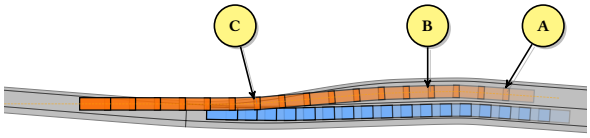


(b) Trajectory of NEVs discovered by the Adversarial search.

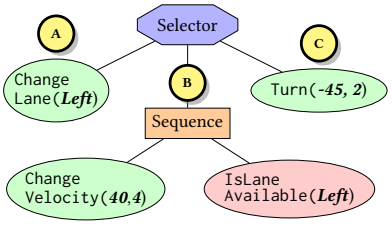


(c) Trajectory of NEVs discovered by the Monte Carlo search.

Fig. 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.

Fig. 9: Example of a Near Miss scenario and corresponding NEV BT discovered by EvoDriver.

D. Use Case: Angled Intersection

Our second use case explores an angled intersection where turning vehicles are required to make sharp turns. AVs are known to exhibit unexpected behavior in such intersections [39], [40], [41]. 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.

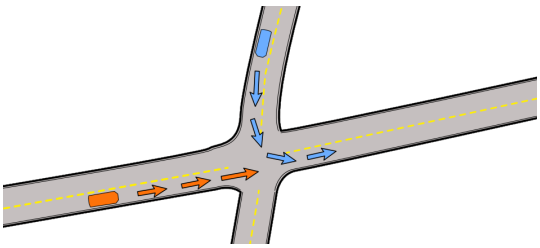


Fig. 10: Overview of the Angled Intersection use case.

a) 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. Specifically, there is only a small window where the NEV and EGO may interact. If the NEV waits too long or passes the intersection too quickly, then the EGO will complete its turn and proceed to the destination without interacting with the NEV. Figure 11 shows the count of test cases discovered for each category by each respective technique. We found results similar to that of the Merge Lane use case, where EVODRIVER discovered more evenly-distributed test cases when compared to the alternative techniques. Additionally, EVODRIVER discovered more test cases in the NEAR MISS category, where NEV behavior resulted in dangerous interactions with the EGO. Table IV 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.

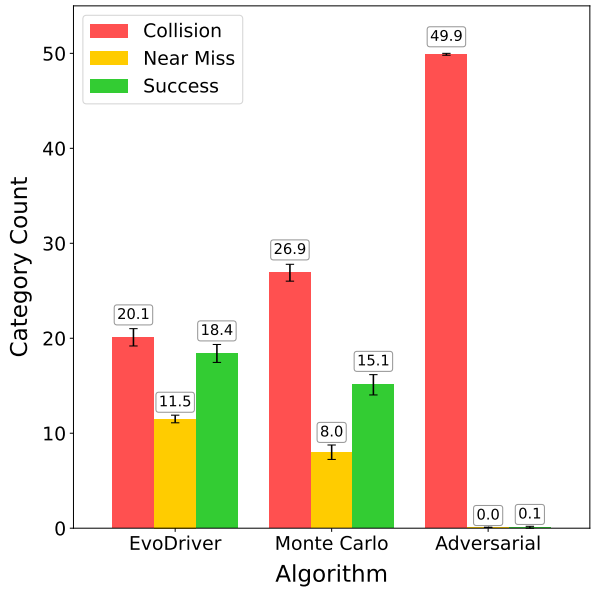


Fig. 11: The inter-category diversity measured for each of the search approaches for Angled Intersection use case.

b) 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. We found similar results to our first use case, where 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**.

TABLE IV: Novelty score statistics for EVODRIVER and alternative approaches for Angled Intersection use case.

Approach	Novelty Score		
	Mean (μ)	Std. Dev. (σ)	p-value (μ_{diff})
EVODRIVER	**0.059**	0.002	-
Adversarial	0.002	0.001	**< 0.01**
Monte Carlo	0.047	0.003	**< 0.01**

c) Analyzing NeV behaviors: Finally, 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. The Adversarial search converged to a set of similar NEV BTs that resulted in similar responses from the EGO and failed to cover all three behavior categories. Likewise, the failures discovered by Monte Carlo search lacked diversity and generally had limited impact on EGO behavior. In contrast, EVODRIVER discovered *diverse test cases* that covered *all* behavior categories.

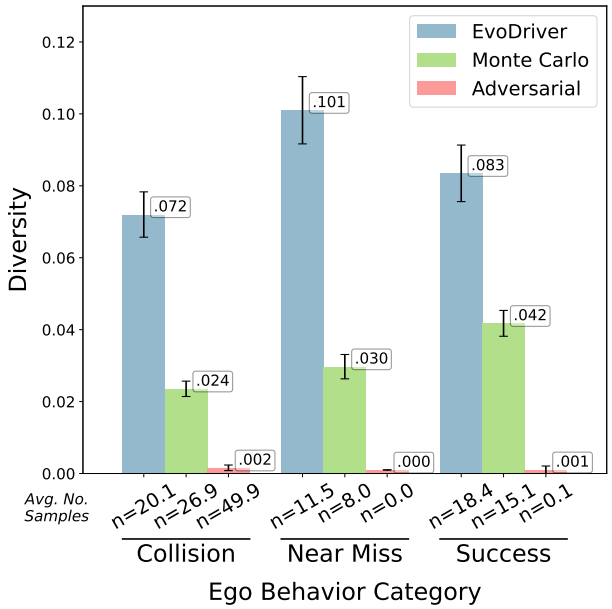
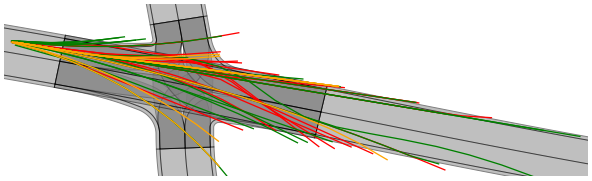


Fig. 12: The intra-category diversity measured for each of the search approaches for Angled Intersection use case.

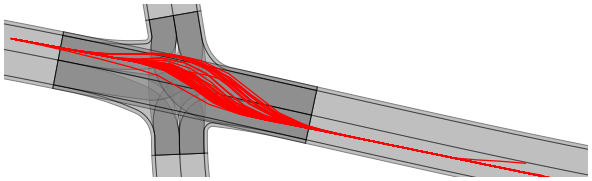
V. DISCUSSION

This section describes our analysis and findings on the results of EVODRIVER. Based on the results from both experiments, we found significant evidence to reject the null hypothesis for both **RQ1** and **RQ2**, and accept the

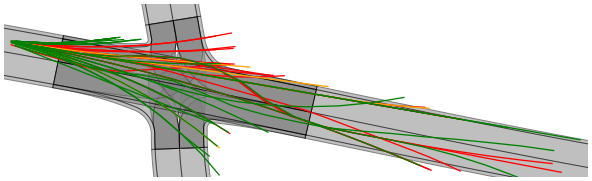
alternative hypotheses that EVODRIVER can find test cases that are more diverse, with respect to both inter- and intra-category diversity. Our post hoc analysis showed that EVODRIVER can find test cases that lead to similar EGO responses as the Adversarial search, in addition to other novel test cases that lead to more diverse near-miss scenarios. We next discuss the benefits of a bottom-up open-ended approach to test case discovery, how the discovered test cases can be used by developers to improve AV robustness, and finally provide insights on the generalizability and computational cost of EVODRIVER.



(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.

Fig. 13: Archive of NEVs discovered by each technique for the Angled Intersection use case.

A. 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 novel 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 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 unique approach to scenario 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 information 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 [20], [21]. EVODRIVER was able to discover a set of increasingly different tests for each category, where each category-specific test suite showed greater diversity than alternative approaches.

B. Towards AV Robustification

This section provides insight into how developers may use the output of EVODRIVER to improve the robustness of AV software. EVODRIVER discovers diverse NEV behaviors that pose uncertainty for the AV, which may lead to uncertain responses from the EGO. This information can be used in different ways to robustify the EGO. First, the 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 [10], [35], [42]. Second, developers can also use the discovered uncertainty to inform revisions and/or additions of new requirements or operating constraints for the AV to robustify against the discovered uncertainty. Finally, we can leverage EVODRIVER-discovered diverse test cases covering three distinct EGO behavior categories to train behavior oracles [35] to support run-time decision-making to detect dangerous external vehicle behavior and trigger automated self-reconfiguration, such as switching to a fail-safe mode.

C. Computational cost

One common limitation of search-based testing is the computation complexity that often hinders rapid test case generation [43]. We found through empirical evidence that individual experiments using EVODRIVER can discover an archive of unique test cases, each leading to different types of behaviors, within 2–3 minutes. This demonstrates the potential feasibility of our work to generate a large collection of test cases in reasonable time for rapid testing in a practical setting. Finally, the computation time may vary across different settings (e.g., computing power, simulation resources, etc.).

VI. RELATED WORK

This section overviews related work relevant to EVODRIVER. First, we discuss related research in AV testing. Next, we overview related work that uses novelty search and GP for assessing AVs. Then, we describe works that leverage search-based approaches for AV testing.

Recent research efforts have addressed the assurance of AVs. McDuff *et al.* [44] proposed CAUSALCITY, a simulator framework that focuses on the explainability of machine learning-based AV models. However, they did not assess AV robustness against uncertainty. Fremont *et al.* [45] introduced SCENIC and demonstrated a formal method approach to scenario-based test generation for AVs. Chan *et al.* [10] 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

diversity of human driving patterns. Zheng *et al.* [46] proposed an optimization-based adversarial testing approach to generate testing scenarios for AVs, but did not explicitly address diversity. Zhong *et al.* [47] proposed a fuzz-based technique to generate scenarios for AV testing using a neural network. 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 [48] demonstrated that GP can be used to evolve BTs, discovering agent controllers using high-level actions, leading to faster convergence. Smith *et al.* [49] showed that phenotypic novelty and objective fitness can be used in combination to produce controllers that adopt multiple strategies. However, their work evolved controllers that exhibited novel behavior in the system under study, rather than controllers whose behaviors resulted in novel responses from a separate system under study. Iovino *et al.* [50] showed that GP can be used to learn the structure of a BT to solve robotic tasks. Montague *et al.* [51] 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 [52], [53] first proposed the use of novelty search, where GPs are used to evolve diverse solutions. Naredo [54] explored a similar approach to evolve solutions for traditional ML tasks such as binary classification or regression analysis. Gomes *et al.* [55], Martinez *et al.* [56], and Velez *et al.* [57] 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.* [14] 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 did not explore agent behaviors. Xie *et al.* [9] proposed a backward target-tree search technique to identify diverse test cases. Betts and Petty [26], Zhou *et al.* [58], and Tian *et al.* [25] presented genetic algorithm-based approaches to generate objective-based test cases for AVs. Huang and Nitschke [59] explored AV cooperation using novelty search, but did not study interactions that can lead to poor AV performance. Langford *et al.* [42] 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 [35] also used novelty search to identify environmental conditions (e.g., raindrops) that can lead to different categories of responses in the ML component. Chan and Cheng [60] applied a similar approach to generate diverse adversarial perturbations for ML components using novelty search. Li *et al.* [61] 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.* [15] proposed a genetic algorithm approach to evolve BTs to discover scenario-based test cases. 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 behaviors in the AV under study.

VII. THREATS TO VALIDITY

This paper harnessed GP with NS 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 (CV) 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" [62]). Finally, we used several use cases (i.e., different road architectures and objectives) to demonstrate EVODRIVER's feasibility, flexibility, and context agnosticism.

VIII. 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 EGO 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 [36], [37], [38], [39], [40], [41]. 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 existing 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 runtime monitoring of external vehicle behavior. Finally, exploring techniques for failure trace analysis of the critical test cases may provide additional insights.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

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

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

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

[25] 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.

[26] 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.

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

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 416–419, 2011.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

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

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

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.