Constrained Co-evolutionary Differential Metamorphic Testing for Autonomous Systems with an Interpretability Approach

HOSSEIN YOUSEFIZADEH*, School of EECS, University of Ottawa, Canada
SHENGHUI GU*, School of EECS, University of Ottawa, Canada
LIONEL C. BRIAND, School of EECS, University of Ottawa, Canada and Lero SFI centre for Software Research, University of Limerick, Ireland
ALI NASR, Waterloo Research Center of Huawei Technologies Canada Co., Ltd., Canada

Autonomous systems, such as autonomous driving systems, evolve rapidly through frequent updates, risking unintended behavioral degradations. Effective system-level testing is challenging due to the vast scenario space, the absence of reliable test oracles, and the need for practically applicable and interpretable test cases. We present *CoCoMagic*, a novel automated test case generation method that combines metamorphic testing, differential testing, and advanced search-based techniques to identify behavioral divergences between versions of autonomous systems. *CoCoMagic* formulates test generation as a constrained cooperative co-evolutionary search, evolving both source scenarios and metamorphic perturbations to maximize differences in violations of predefined metamorphic relations across versions. Constraints and population initialization strategies guide the search toward realistic, relevant scenarios. An integrated interpretability approach aids in diagnosing the root causes of divergences. We evaluate *CoCoMagic* on an end-to-end autonomous driving system, InterFuser, within the Carla virtual simulator. Results show significant improvements over baseline search methods, identifying more distinct high-severity behavioral differences while maintaining scenario realism. The interpretability approach provides actionable insights for developers, supporting targeted debugging and safety assessment. *CoCoMagic* offers an efficient, effective, and interpretable way for the differential testing of evolving autonomous systems across versions.

CCS Concepts: • **Software and its engineering → Search-based software engineering**.

Additional Key Words and Phrases: autonomous systems, autonomous driving systems, system-level testing, metamorphic testing, differential testing, search-based testing, automated testing, cooperative co-evolutionary algorithms

1 Introduction

Autonomous Systems (ASs) are complex software-hardware systems designed to autonomously make real-time operating decisions in open contexts by processing data from a multitude of inputs. These systems are mostly built with Artificial Intelligence (AI) units that learn from vast datasets and may, in the future, continuously evolve through reinforcement learning. As ASs continue to advance, the need for systematic testing becomes increasingly important, to maintain public trust and to satisfy regulatory requirements [43, 53, 75]. Concurrently,

*Hossein Yousefizadeh and Shenghui Gu contributed equally to this work.

Authors' Contact Information: Hossein Yousefizadeh, hyous028@uottawa.ca, School of EECS, University of Ottawa, Ottawa, Ontario, Canada; Shenghui Gu, sgu2@uottawa.ca, School of EECS, University of Ottawa, Ottawa, Ontario, Canada; Lionel C. Briand, lbriand@uottawa.ca, School of EECS, University of Ottawa, Ottawa, Ontario, Canada and Lero SFI centre for Software Research, University of Limerick, Limerick, Ireland; Ali Nasr, ali.nasr2@huawei.com, Waterloo Research Center of Huawei Technologies Canada Co., Ltd., Waterloo, Ontario, Canada.

ACM 1557-7392/2026/6-ART
https://doi.org/10.1145/3821438

AS development relies on iterative refinement, incorporating new real-world data, optimized network architectures, hyperparameter tuning, and algorithmic tweaks to boost AS performance across diverse operating conditions [49, 85, 88]. However, each update can inadvertently introduce changes that undermine previously validated behaviors or functionalities. Assumptions about system behavior or responses that once appeared benign may no longer hold under new or even familiar conditions, leading to unsafe outcomes. Without systematic testing, such behavioral degradations can remain hidden, compromising system reliability and safety. Consequently, there is an urgent need for effective testing methods that systematically identify critical violations triggered by AS updates, particularly as modern ASs are increasingly built on complex AI models whose behavior can be difficult to predict and verify.

Meeting this need is complicated by the fact that the behavior of ASs across diverse scenarios is both complex and unpredictable, making it challenging to detect degradations and trace their root causes. Variations in environmental conditions, task parameters, and system inputs interact with the uncertainty inherent in Machine Learning (ML) or Deep Learning (DL) models to produce a vast array of possible AS responses. In practice, a single update, whether to the model, the control policy, or the training data, can affect hundreds of interconnected components. Consequently, a failure observed in one scenario, whether a safety violation or a subtle or indirect unsafe behavior, may arise from a subtle modification elsewhere. Isolating which change triggered the degradation, therefore, requires re-creating and re-evaluating numerous complex test cases, a process that demands extensive computational resources and careful methodological design.

In addition, the main obstacle in testing ASs is the lack of dependable test oracles. This challenge arises because, in many cases, especially in complex or open-ended scenarios, formal specifications or expected outcomes are either absent or difficult to establish [25, 67]. Unlike traditional software, where a function's input and output can be precisely defined, ASs often operate in environments where a spectrum of behaviors may be acceptable for a given situation. For instance, a robot navigating a crowded space may choose different but equally valid paths, or an automated control system may apply varying control signals within a safe operating range. Human experts sometimes disagree on the correct maneuver in novel situations, and encoding every nuance of safe behavior into a fixed rule set is generally infeasible. Without clear pass/fail criteria, automated testing cannot definitively flag deviations as degradations, making it nearly impossible to scale testing across the full range of real-world operating conditions.

Furthermore, even when degradations can be detected without a formal oracle, the practical relevance of such detections depends heavily on the realism of the scenarios in which they occur. While artificially constructed edge cases can expose system weaknesses, failures observed in scenarios that closely resemble conditions the system has recently or commonly encountered during real-world operation are often of greater practical importance. Such scenarios are inherently more likely to increase the probability that any detected failure will manifest after deployment. We refer to such scenarios as "realistic" throughout this paper. Therefore, prioritizing failure discovery in realistic settings enhances the practical relevance of the testing process and helps focus engineering efforts on risks more likely to impact users.

To address these challenges, we propose *Cooperative Co-evolutionary differential and Metamorphic Automated Generator for Interpretable Cases in autonomous systems (CoCoMagic)*, an effective and efficient automated test case generation method that integrates Differential Testing (DT) with Metamorphic Testing (MT) [13] to expose behavioral divergences between AS versions. In *CoCoMagic*, DT serves as the primary mechanism for efficiently detecting diverse instances of behavioral divergence by comparing the outputs of two ASs under identical test inputs. However, DT alone cannot address the oracle problem, as it does not assess the impact of detected behavioral differences on system safety. MT is therefore integrated into *CoCoMagic* to provide a mechanism for assessing the safety implications of such behavioral changes. MT provides a practical approach to evaluating system behavior using Metamorphic Relations (MRs), which define expected relationships between related inputs and outputs rather than verifying the absolute correctness of individual test outputs. As a result, it is particularly well-suited for domains such as autonomous driving, where correct behavior is often context-dependent and ambiguous [75, 92], and where labeled test data may be impractical or too expensive. Furthermore, unlike threshold-based safety metrics, which can only detect specific types of safety violations, MRs can capture a broader range of unsafe behaviors, including subtle and indirect ones that do not necessarily lead to immediate safety violations but indicate underlying instability or degradation in system performance that could eventually compromise safety.

In the context of evolving ASs, where system updates may subtly alter behavior, DT coupled with MT can help reveal deviations that violate the expected output relations defined by MRs. Such violations may indicate degradations or unexpected side effects introduced during model fine-tuning or other updates. We cast test case generation as a search problem and seek inputs in which the extent of violation of the expected output relation differs markedly across versions. This paired formulation is important because the goal is not merely to find severe violations in either version, but to identify test conditions under which the update changes the system's safety-relevant behavior. By systematically identifying such inputs, our method yields effective yet diverse test cases that highlight safety risks and unintended behaviors introduced by the continuous evolution of ASs. We also incorporate constraints and a population initialization strategy to guide the search toward realistic test cases that reflect real-world conditions. Specifically, the population is initialized with scenarios derived from real-world execution data, providing a realistic starting point, while a penalty term in the fitness function discourages the generation of test cases that deviate significantly from observed operational conditions. This design ensures that the generated cases are relevant and applicable to practical scenarios, particularly in contexts where the updated system is expected to operate. The extent to which realism is prioritized can also be adjusted, allowing practitioners to balance realism against other testing objectives based on their specific needs.

To further enhance the utility of *CoCoMagic*, we integrate an interpretability approach to support the analysis and understanding of the root causes of the identified behavioral divergences. This approach allows us to analyze the generated test cases and gain insights into the underlying factors contributing to the observed discrepancies, thereby helping developers address potential safety risks more effectively, producing cases that are not only effective at exposing divergences but also understandable and interpretable.

While our method is designed for general applicability to ASs operating in open and complex environments, this work focuses on Autonomous Driving Systems (ADSs) as a representative and challenging case study. End-to-end ADSs exemplify one of the most complex forms of ASs, positioned at the higher end of the complexity spectrum [10]. To evaluate the effectiveness and efficiency of *CoCoMagic*, we conducted experiments using the Carla simulator [21] with the InterFuser ADS [73]. The results show that *CoCoMagic* significantly outperforms baseline methods in identifying severe behavioral discrepancies, with consistent advantages across a wide range of search budgets and configurations. This superior performance highlights *CoCoMagic* as a promising and scalable solution for testing ADSs. The use of constraints and population initialization effectively guides the search toward realistic and practically relevant test cases, thereby enhancing the applicability of the generated test cases. Moreover, the interpretability approach integrated into *CoCoMagic* enables a detailed analysis of behavioral divergences, offering valuable insights into the underlying factors contributing to the observed discrepancies.

The major contributions of this paper are summarized as follows:

- We propose *CoCoMagic*, a novel method for automated test case generation that combines MT and DT to expose behavioral divergences between different versions of ASs. We also introduce constraints and a population initialization strategy to guide the search toward realistic test cases that reflect real-world conditions.
- We incorporate an interpretability approach into *CoCoMagic* to facilitate the analysis and understanding of the root causes behind the identified behavioral divergences.

- We apply *CoCoMagic* to a complex case study utilizing an industry-grade simulator and a high-performing ADS.
- We provide a comprehensive evaluation of *CoCoMagic* through large-scale experiments and comparisons against baseline methods, showing its ability to identify behavioral discrepancies and highlight potential safety risks introduced by updates.

CoCoMagic builds on our previous method, *CoCoMEGA* [89], and mainly differs from it by focusing on efficiently and effectively testing and analyzing updates across system versions. The key differences are as follows:

- *CoCoMagic* is specifically designed to systematically detect behavioral divergences between different versions of ASs, rather than focusing solely on testing a single version.
- It incorporates constraints and a population initialization strategy to guide the search toward realistic test cases that reflect real-world conditions.
- It integrates an interpretability approach that facilitates understanding of the root causes of observed behavioral divergences by analyzing the generated test cases.

The remainder of this paper is organized as follows. Section 2 introduces the necessary background concepts. Section 3 formally defines the problem and outlines the key challenges. Section 4 reviews related research and highlights existing gaps. Section 5 presents a detailed description of *CoCoMagic*. Section 6 details the design of our empirical evaluation. Section 7 presents and interprets the experimental results. Section 8 analyzes the findings and discusses potential threats to validity. Finally, Section 9 summarizes the paper and outlines future work directions.

2 Background

In this section, we provide background information on the key concepts and our previous work that underpin our proposed framework for testing evolving ASs.

2.1 Metamorphic Testing

Metamorphic Testing (MT), first introduced by Chen et al. [12], addresses the oracle problem in software testing by leveraging MRs. Instead of requiring a ground-truth output for every test case, MT begins with a set of source test inputs and their observed outputs, then applies transformations to generate new follow-up inputs. The expected relationship between the outputs of the original and transformed inputs is defined by an MR. If the program's outputs violate this relationship, a defect is detected. For example, consider testing a program f intended to compute $\sin(x)$. Normally, one can check the correctness directly by verifying if $f(x) = \sin(x)$ for any input x. However, assume that the ground-truth value $\sin(x)$ is unknown, then such direct checking is not possible. MT provides an alternative approach by testing the program using an MR. For instance, using the mathematical property $\sin(x) = \sin(x + 2\pi)$, we can define an MR stating that $f(x)$ should equal $f(x + 2\pi)$. Testing then reduces to checking whether this relation holds for any input x. To ground these ideas more formally, we now introduce the precise definition of an MR and explain how it is represented in our framework.

An MR generally specifies a relation $\mathcal{R}$ that defines a relationship between a sequence of inputs and their respective outputs [13]. A common category of MRs consists of two components: an input relation *ir* and an output relation *or*. The input relation *ir* specifies how two inputs x_1 and x_2 are related, while the output relation *or* describes the expected relationship between their outputs $f(x_1)$ and $f(x_2)$ [20], where f represents the function or system under test. The relation $\mathcal{R}$ holds when both *ir* and *or* are satisfied, and can be formally expressed as:

$$\mathcal{R}\,(x_1, x_2, f(x_1), f(x_2)) \coloneqq ir\,(x_1, x_2) \rightarrow or\,(f(x_1), f(x_2))$$

This formulation allows an MR to be interpreted as a consistency rule that governs how the system's output should change in response to a defined transformation of its input. In our work, we formally define an MR as a tuple $mr \coloneqq (ir, or)$. Given the input relation ir providing an abstract specification of how a source test input can be transformed, we can derive concrete metamorphic transformations that modify the attributes of a source input to produce a follow-up input. A violation occurs when the follow-up input satisfies ir, but the corresponding outputs fail to satisfy or, indicating that the MR has been violated. In our search framework, we employ a set of MRs that share a common output relation or to guide the search, which can be defined as $MR \coloneqq \{mr_i = (ir_i, or) \mid i = 1, \ldots, k\}$, where k denotes the number of MRs in the set.

2.2 Differential Testing

Differential Testing (DT) is a testing technique in which multiple comparable implementations of a system are exercised on the same automatically generated inputs, and any divergence in their outputs is treated as evidence of a potential bug [61]. In practice, DT works by executing large numbers of inputs across multiple implementations and automatically flagging discrepant executions for analysis. DT has been successfully applied to a variety of software systems [50, 65, 86], whenever multiple independent implementations or versions are available for comparison on shared inputs.

DT is well suited to iterative development workflows in which models, training data, and control policies are continually revised. By running the same test inputs across successive versions of an AS and comparing their behavior, it directly targets update-induced changes, surfacing cases where system behavior improves or degrades. This comparison enables efficient identification of degradations and other unintended behavioral changes across system updates.

2.3 Cooperative Co-evolutionary Algorithms

Cooperative Co-Evolutionary Algorithms (CCEAs) form a particular type of Evolutionary Algorithms (EAs), a family of algorithms inspired by biological evolution and widely used to solve optimization problems that are difficult to handle with standard mathematical optimization techniques [54]. In EAs, candidate solutions are represented as individuals within a population, and their quality is evaluated by a fitness function. Individuals with higher fitness are more likely to be selected as parents, and new individuals are generated through crossover (combining parts of parent solutions) and mutation (introducing random variation in an individual). Through iterative cycles of selection, crossover, and mutation, the population evolves toward increasingly effective solutions.

However, many real-world problems involve extremely high-dimensional search spaces, making standard EAs inefficient [54, 56]. To address this challenge, CCEAs, originally proposed by Potter and De Jong [66], decomposes the original problem into a set of lower-dimensional and tractable subproblems, each of which can be solved in a separately evolving subpopulation as in conventional EAs. As individuals from each subpopulation must work together to form a complete solution, an individual's fitness is evaluated based on the joint fitness of that complete solution, which is created by pairing the individual with representative collaborators from other populations or from population archives such as those used in *iCCEA* [63]. By leveraging decomposition and collaborative evaluation, CCEAs can effectively search high-dimensional spaces, making them well-suited to our task of identifying scenarios in a complex, open-ended environment.

2.4 RuleFit

The *RuleFit* algorithm, introduced by Friedman and Popescu [27], combines decision trees with linear models to construct predictive models that are both expressive and interpretable. Linear regression models are simple and interpretable, but cannot naturally capture feature interactions or nonlinear relationships. In contrast, decision trees can model complex interactions, yet they become increasingly difficult to interpret as their depth and number of leaf nodes grow. *RuleFit* bridges this gap by learning a sparse linear model that incorporates both the original features and a set of automatically generated decision rules. These rules are derived from decision trees and explicitly encode feature interactions. Each path from the root to a leaf in a tree is transformed into a binary rule by conjoining the split conditions along that path. The trees themselves are trained to predict the target outcome, ensuring that the generated rules are relevant to the learning task. *RuleFit* is flexible with respect to the tree-generation process, and any method that produces an ensemble of trees, such as Random Forests (RFs) or Gradient Boosted Decision Trees (GBDTs), can be used. The resulting rules are then combined with the original features in a sparse linear model, typically using Lasso [77], which promotes sparsity and selects a compact set of informative rules.

To illustrate, consider the example from the original *RuleFit* paper [27], which predicts the median house value of a Boston neighborhood. The input features include the average number of rooms per dwelling, the proportion of owner-occupied units built before 1940, and weighted distances to five Boston employment centers, while the output is the median house value in thousands of dollars. One rule generated by *RuleFit* is: "IF the number of rooms > 6.64 AND the concentration of nitric oxides < 0.67, THEN increase the predicted median house value by 1.30". This rule captures an interaction between two features that a standard linear model would not represent without manual specification. By automatically introducing interaction terms, *RuleFit* alleviates the need to manually engineer feature interactions and improves linear models' ability to capture nonlinear effects. At the same time, the resulting model remains interpretable, as each rule is a simple binary condition indicating whether it applies to a given instance. *RuleFit* supports both regression and classification tasks, making it a versatile tool for predictive modeling where interpretability and expressiveness are equally important. In our framework, we leverage *RuleFit* to derive such interpretable rules from the identified test cases that expose the behavioral differences between ASs. These rules help engineers understand and communicate the specific scenario conditions under which an updated AS exhibits different behavior from a previous version, thereby facilitating debugging and improvement efforts.

2.5 CoCoMEGA

CoCoMEGA [89] is a search-based testing framework that combines MT with an archive-based CCEA to generate system-level test cases for ASs. The key idea is to decompose the high-dimensional test generation problem into two coevolving populations: one representing source scenarios and the other representing perturbations derived from a predefined set of MRs. Each perturbation encodes a concrete metamorphic transformation (e.g., adding a pedestrian, changing weather, or adjusting vehicle speed) that can be applied to a source scenario to produce a follow-up scenario. By collaborating individuals from the two populations, *CoCoMEGA* forms complete solutions (i.e., scenario-perturbation pairs) that are executed in simulation to assess the extent to which the corresponding MR is violated. In the following, we summarize the key concepts and components of *CoCoMEGA*.

2.5.1 Representations. There are two co-evolving populations involved in *CoCoMEGA*, i.e., one for source scenarios and one for perturbations.

A scenario s can be formally represented as a vector composed of real and integer values. In practice, the specific vectorization scheme depends (in part) on the observable and controllable elements of the underlying system or simulator. For example, in the context of ADSs, a scenario typically includes the ego vehicle, its trajectory, environmental factors (e.g., weather), and both dynamic (e.g., other vehicles and pedestrians) and static (e.g., obstacles) objects [33, 74]. Each of these elements is characterized by multiple attributes of different types, including continuous values and categorical variables encoded as integers. In this context, let *ego* be the ego vehicle and its attributes, *WP* be a set of waypoints that characterizes the trajectory, *ATTR* be a set of global attributes, e.g., weather and brightness, *STAT* be a set of static objects, e.g., traffic signs, and *DYNA* be a set of dynamic objects, e.g., pedestrians and vehicles. The representation of a scenario can be defined as the tuple $s \coloneqq (ego, WP, ATTR, STAT, DYNA)$, where:

$$
\begin{aligned}
ego &\coloneqq (mod_{ego}, pos_{ego}, rot_{ego}, v_{ego})\\
WP &\coloneqq \{wp_i \mid i \in \mathbb{N}^*\}\\
ATTR &\coloneqq \{attr_i \mid i \in \mathbb{N}^*\}\\
STAT &\coloneqq \{stat_i \coloneqq (mod_i, pos_i, rot_i) \mid i \in \mathbb{N}^*\}\\
DYNA &\coloneqq \{dyna_i \coloneqq (mod_i, pos_i, rot_i, v_i) \mid i \in \mathbb{N}^*\}.
\end{aligned}
$$

Here, *mod* represents the type of the object, e.g., the model of the vehicle such as a motorcycle or a car, *pos* represents the position of the object in three-dimensional space, *rot* represents the direction the object is facing in three-dimensional space, and v represents the speed of the respective objects. Each attribute (weather, brightness, object speed, etc.) is bounded by predefined ranges that reflect realistic conditions and conform to Carla's valid parameters. These ranges are also configurable, allowing testers to tailor the space of possible values to suit different testing requirements or fidelity levels.

A perturbation can be formally defined as a sequence of metamorphic transformations, denoted as $q \coloneqq \langle c_1, c_2, \ldots, c_k \rangle$, which are applied sequentially to a source scenario s to produce a follow-up scenario $q(s)$. Each transformation c_i in q corresponds to a unique input relation ir_i from the given set of MRs. Typically, a transformation c_i may represent a modification to the source scenario, such as adding a new object, removing an existing one, or replacing an existing object with a new one. It may also represent no change at all, with no modification applied to that specific transformation. To ensure realism, every transformation must remain within the predefined allowable ranges of the involved attributes. This flexible structure allows perturbations to be constructed selectively, so that not every available transformation needs to be included. However, we require that at least one actual transformation be applied to produce a follow-up scenario with meaningful variation.

2.5.2 Archive Strategy. To avoid premature convergence and encourage the exploration of diverse failures, *CoCoMEGA* maintains archives of scenarios, perturbations, and complete solutions, and relies on a heterogeneous distance metric [83] over solutions to select archive members that are both high-fitness and well spread in the search space.

The heterogeneous distance used in *CoCoMEGA* provides a unified way to measure how different two individuals (i.e., scenarios or perturbations) are, even though scenarios contain multiple types of elements and attributes. At the attribute level, numerical values such as position or speed contribute to a normalized difference. In contrast, categorical values, such as object type, contribute a small fixed penalty when mismatched and zero when identical. These attribute-level comparisons allow us to measure the similarity between two objects. To compare sets of objects, such as static or dynamic objects in a scenario, the distance pairs objects across the two sets according to their attribute-level distances and aggregates the resulting best-match distances, where each best match corresponds to the minimum distance achievable by pairing an object with its most similar counterpart in the other set, capturing structural differences even when the two scenarios contain different numbers of objects. Finally, the distances for static objects, dynamic objects, and global attributes are combined into a single value, yielding an overall measure of scenario similarity that accounts for variations in objects, attributes, and environmental conditions. This unified metric enables consistent comparison across complex, heterogeneous scenarios commonly found in ADS testing.

The archive strategy is designed around the heterogeneous distance and maintains a balance between exploiting high-fitness solutions and exploring diverse regions of the search space. At each generation, the algorithm begins by inserting the individual with the highest fitness into the archive, thereby preserving strong solutions. It then incrementally fills the remaining archive slots by repeatedly selecting the individual that maximizes the increase in overall diversity when added to the current archive, encouraging the search to cover a broad range of behaviors and reducing the likelihood of premature convergence. Diversity is quantified using the Pure Diversity metric [78], which evaluates the spread of individuals to capture both primary dissimilarities and the overall extent of coverage in the search space, resulting in an archive that both retains high-fitness individuals and fosters exploration.

2.5.3 Genetic Operators. Standard evolutionary operators (selection, crossover, mutation) are applied separately to the two populations, which are periodically merged with their corresponding archives, yielding an iterative co-evolutionary process that gradually accumulates a set of distinct, high-severity violating test cases. Since the complexity of the scenario and perturbation representations precludes the use of simple genetic operators, *CoCoMEGA* employs specialized operators tailored to each population, as described below.

Selection. The selection operator for both populations uses standard tournament selection, the most prevalent method in evolutionary algorithms [79]. Candidate individuals are selected for reproduction based on their relative fitness rankings.

Crossover. The crossover operator for scenarios creates new offspring by combining characteristics from two parent scenarios while preserving their essential structure. The parameters of the ego vehicle *ego* and the trajectory *WP* remain unchanged, while global attributes *ATTR*, static objects *STAT*, and dynamic objects *DYNA* are exchanged between parents using a uniform crossover. This approach allows offspring to inherit traits from both parents, such as weather conditions, vehicle positions, and obstacle configurations, while maintaining the overall integrity of the scenario.

For perturbations, the crossover operator works pairwise across sequences of metamorphic transformations from two parents. Each corresponding transformation is combined using uniform crossover, which may swap parameters when both transformations affect the same type of element, such as adding vehicles or modifying global attributes. If the paired transformations modify different types of elements, they are preserved as is. This mechanism enables new perturbation sequences that inherit traits from both parents while maintaining valid and meaningful modifications to the scenario.

Mutation. The mutation operator for scenarios introduces variations by modifying global attributes *ATTR* and the properties of static objects *STAT* and dynamic objects *DYNA* while keeping the ego vehicle *ego* and trajectory *WP* unchanged. Numerical attributes are perturbed using polynomial mutation [18], and categorical attributes are altered using integer randomization [54]. Additionally, the operator can incrementally add or remove objects from the scenario according to a probabilistic scheme, helping explore a broader range of conditions and preventing the scenario from growing excessively over generations. These changes generate subtle but meaningful variations in scenarios, such as adjusting weather, brightness, vehicle positions, or pedestrian behaviors, while preserving the overall structure.

For perturbations, mutation is applied to each metamorphic transformation in a sequence, using the same numerical and categorical mutation strategies as for scenarios. Depending on the type of transformation, the mutation may assign a new value to a global attribute, alter parameters of static or dynamic objects, or enable or disable a transformation entirely. This approach ensures that offspring perturbations remain valid while introducing diversity, producing a rich set of variations that can explore different potential outcomes in the scenarios.

2.5.4 How CoCoMagic Extends CoCoMEGA for AS Testing. *CoCoMEGA* provides a first step towards co-evolutionary MT. Still, it does not fully address several demands that arise in the development and testing of real-world ASs. CoCoMagic builds directly on *CoCoMEGA* and extends it along the following four key dimensions.

Differential Testing. CoCoMEGA focuses on testing a single system under test by maximizing the degree of MR violation for that system. In practice, however, engineers frequently need to compare multiple versions or configurations of an AS (e.g., before and after a software update, or across alternative controller designs) and detect degradations or unexpected behavioral divergence. A key motivation for CoCoMagic is that behavioral divergence is a paired property of two system versions under the same test condition. Running a single-version testing method independently on each version may find severe MR violations, but it does not directly optimize for cases where the degree of violation changes across versions. CoCoMagic therefore evaluates each generated scenario–perturbation pair on both versions and uses the resulting difference in MR violation as the search objective, enabling a controlled search for update-induced behavioral changes.

Constrained Search Space. CoCoMEGA performs evolutionary search without explicitly enforcing constraints that ensure the realism of generated scenarios, so many candidates can fall outside the system's operational environment or correspond to implausible traffic situations. This not only reduces the usefulness of the discovered failures but also wastes simulation budget on cases that engineers are unlikely to encounter in practice. In contrast, *CoCoMagic* constrains the search to remain close to a diverse archive of realistic scenarios derived from previous AS executions. This steers the search towards realistic, operationally plausible scenarios that are more likely to arise in the operational environment.

Scenario Initialization. CoCoMEGA initializes its populations largely at random, which is appropriate for generic search but does not exploit information accumulated during iterative development and testing. *CoCoMagic* introduces *Runtime Initialization (RI)* strategies that bias the initial population towards regions of the scenario space that are close to previously detected failures. This improves search efficiency, accelerates the discovery of problematic scenarios, and better reflects how engineers iteratively test evolving AS versions.

Interpretability Layer. The output of *CoCoMEGA* is a set of high-dimensional solutions with associated fitness values, leaving it to engineers to manually infer why certain regions of the scenario space are particularly problematic. *CoCoMagic* adds an interpretability component, which learns rule-based models that approximate the relationship between scenario attributes and MR violation outcomes. These rules provide human-readable explanations of the form "if a combination of conditions on scenario parameters holds, then the likelihood or magnitude of violation is high". Such explanations help engineers understand and communicate why the AS fails, support debugging and design decisions, and make the testing outcomes more actionable in industrial settings.

Summary. Overall, *CoCoMagic* extends *CoCoMEGA* from a general-purpose co-evolutionary MT engine to a practice-oriented framework that supports DT, respects domain constraints, leverages prior knowledge, and produces interpretable insights tailored to real-world AS development.

3 Problem and Challenges

In this section, we first present a motivating example to illustrate the context of our research. Next, we formally define the problem we aim to solve. Finally, we outline the key challenges that arise in this context and the need for a systematic approach to address these challenges.

3.1 Motivating Example

Consider an ADS that has been fine-tuned to operate in a new urban environment with different road layouts and traffic patterns. The original version of the system was primarily trained and tested in suburban settings, where traffic is relatively light and road structures are simpler. However, the updated version has undergone fine-tuning to better handle the complexities of urban driving, including navigating dense traffic, frequent stops, and complex intersections. After fine-tuning, it is essential to assess how the system's behavior has changed and whether these changes translate into improved safety and performance in urban conditions.

Evaluating such changes is challenging. First, determining whether the updated ADS behaves correctly in complex urban scenarios is difficult, as there is often no clear test oracle specifying the expected outcome for every situation. For example, when encountering multiple pedestrians crossing at different speeds in a crowded street, it is nontrivial to define precise ground-truth behaviors against which the system's decisions can be judged. Second, the generated test scenarios must remain representative of the new urban environment. Testing the system with unrealistic or overly synthetic scenarios may fail to reveal meaningful behavioral differences that arise in real city driving. Moreover, when behavioral differences are detected between the original and updated systems, engineers need a way to understand why these differences occur. Simply identifying failing or divergent scenarios is insufficient without interpretable explanations that reveal which environmental factors or interactions contribute to the observed changes. Finally, this evaluation process should be efficient enough to be performed regularly, especially if the ADS undergoes frequent updates or fine-tuning.

3.2 Problem Definition

This motivating example highlights the need for an automated, effective, and time-efficient system-level testing methodology that systematically identifies scenarios in which the behavior of an updated AS has improved or declined relative to the original version, thereby supporting repeated, continuous testing throughout the system's development lifecycle.

Specifically, we re-express the problem as a search problem. Let $\mathcal{S}$ denote the space of all possible operating scenarios, and let $\mathcal{Q}$ denote the space of all possible perturbations derived from a set of MRs that share the same output relation *or*. We define a function $\Delta E_{or}(s, q)$ that quantifies the difference in the extent of MR violation of *or* between two versions of an AS, given a source scenario $s \in \mathcal{S}$ and its follow-up scenario $q(s)$, which results from applying perturbation $q \in \mathcal{Q}$ to s. For illustration, consider a testing context where the goal is to evaluate how different versions of an ADS behave when encountering a pedestrian under various weather conditions. The space $\mathcal{S}$ includes all possible scenarios defined by factors such as road geometry, traffic density, and initial pedestrian placement. The space $\mathcal{Q}$ includes possible perturbations, such as changes in weather (e.g., fog or rain), derived from an MR expecting the ADS to reduce speed when visibility decreases. The specific factors and their value ranges defining $\mathcal{S}$ and $\mathcal{Q}$ depend on the subject system and the simulation environment in which it operates. Now, given a source scenario $s \in \mathcal{S}$ with clear visibility, applying a perturbation $q \in \mathcal{Q}$ yields a follow-up scenario $q(s)$ with reduced visibility. The function $\Delta E_{or}(s, q)$ then measures how inconsistently the two ADS versions respond to the same visibility change, specifically quantifying the difference in the extent to which they violate the expected speed reduction behavior defined by *or*. A larger value indicates greater inconsistency between system versions in meeting the expected MRs.

Definition 1 (Problem). The problem is to find a set $\mathcal{SP}$ of scenario-perturbation pairs $(s, q) \in \mathcal{S} \times \mathcal{Q}$ such that the extent of violation of the expected output relation *or* differs between two versions of the AS when applied to the same perturbed scenario $q(s)$:

$$\mathcal{SP} = \{(s, q) \in \mathcal{S} \times \mathcal{Q} \mid \Delta E_{or}(s, q) > 0\}$$

3.3 Challenges

However, assessing the impact of updates presents several challenges:

- **Scalability and Coverage.** Thoroughly testing ASs in all conditions is infeasible due to the vast space of possible execution scenarios. Many factors influence AS behavior, such as road conditions, weather, traffic dynamics, and pedestrian interactions in the case of ADSs. Ensuring that the testing framework effectively captures the full range of behavioral changes between the original and updated versions is a significant challenge. Without comprehensive coverage, some critical failure cases may go undetected, posing safety risks.

- **Oracle Problem.** Determining whether a behavioral change in the AS constitutes an improvement or a decline is non-trivial. For example, tasks in ADSs, including perception, planning, and decision-making, are highly complex, and their correctness cannot usually be explicitly and precisely defined. The unpredictable nature of real-world traffic conditions further complicates this problem, making it difficult to establish ground-truth expectations for every possible scenario. The absence of well-defined test oracles can lead to ineffective evaluations, as it becomes challenging to distinguish between beneficial adaptations and unintended degradations.
- **Scenario Realism.** While artificially constructed edge cases can reveal vulnerabilities, failures found in realistic scenarios, particularly those closely resembling situations already encountered during system execution, are often of greater practical importance. Such scenarios are inherently more likely in deployment, making any detected failure pose a higher operational risk. Therefore, guiding the testing process toward realistic conditions enhances the relevance of test results, ensuring that identified violations reflect risks that are plausible in practice rather than purely theoretical.
- **Interpretability.** ASs rely on ML or DL-based components with a vast number of parameters and intricate decision-making mechanisms. Fine-tuning involves modifying these parameters to optimize performance, but understanding how these modifications influence overall system behavior is not straightforward. Without systematic, well-designed testing methodologies, it is difficult to diagnose issues and ensure that system changes align with expected improvements.
- **Time Constraints.** In many scenarios, fine-tuning occurs automatically and frequently, particularly in self-adaptive systems that continuously refine their models. Given the rapid pace of such updates, an efficient testing process is essential. The challenge is to develop an automated, fast testing approach that provides insights into AS behavioral changes without significantly delaying deployment.

Addressing these challenges is critical to ensuring the safety of updated ASs. A robust testing methodology should efficiently identify behavioral changes, overcome the oracle problem, enable interpretability, and operate within realistic time constraints.

4 Related Work

This section reviews prior work in two main areas that intersect with our contributions, namely the Search-based Testing of ASs and the Metamorphic Testing of ASs. We focus on prior work that targets system-level testing of ASs, including ADSs, drones, and robots, where tests are generated or prioritized using search-based optimization, fuzzing, or MT in simulation. We exclude studies that address only isolated modules (such as perception components) or rely solely on random test generation without feedback from system behavior. Within this scope, we prioritize approaches that aim to uncover safety-critical behaviors and introduce design elements closely related to our method.

4.1 Search-based Testing of Autonomous Systems

Search-based testing has emerged as a key strategy for the system-level evaluation of ASs across a range of domains, including autonomous vehicles, aerial drones, robotic platforms, and other complex multi-input AI-based systems. The central idea is to automate the exploration of complex, high-dimensional scenario spaces using evolutionary search to uncover failure-inducing behaviors.

Ben Abdessalem et al. [1, 2] pioneered a multi-objective strategy that combines search-based testing with surrogate models based on neural networks to assess advanced driver assistance systems in simulation. Dreossi et al. [23] proposed a compositional, search-based framework tailored for ML-enabled ADSs, leveraging constraints from both the perception and system input spaces to efficiently isolate counterexamples. Sartori [70] proposed a simulation-based testing framework for autonomous robots that leverages search-based techniques to guide the generation and selection of virtual test worlds, incorporating test objectives, prior results, and dynamic agents. Haq et al. [32] introduced *SAMOTA*, which employs many-objective optimization with surrogate models to efficiently surface safety violations by approximating expensive simulations. Kolb et al. [44] developed a set of fitness function templates tailored for search-based testing of ADSs in intersection scenarios. This work builds on earlier efforts in highway environments [34] and demonstrates that tailored fitness definitions can significantly enhance the generation of diverse, safety-relevant tests compared to random testing baselines. Luo et al. [55] presented *EMOOD*, a search-based framework that leverages evolutionary strategies to systematically uncover test scenarios involving compound violations of system requirements. Birchler et al. [6, 7] proposed test case prioritization strategies aimed at optimizing the fault detection efficiency in regression testing. Their empirical results demonstrated that their approach can enhance early fault detection and testing efficiency relative to baseline methods.

Fuzzing techniques have also been employed to trigger faulty behaviors by systematically perturbing operating scenarios. Originally developed for software testing, fuzzing is the automated, repeated mutation of inputs to explore a wide range of system behaviors and increase the likelihood of exposing faults. Li et al. [47] introduced *AV-Fuzzer*, which leverages genetic algorithms to perturb traffic participants in the Apollo platform [4], leading to broader discovery of critical safety violations compared to methods such as random fuzzing or adaptive stress testing. Cheng et al. [14] proposed *BehAVExplor*, a fuzzing approach that uses unsupervised clustering to model ego vehicle behavior and guide scenario generation based on behavior diversity and violation potential. Evaluated on Apollo and LGSVL [68], it outperformed state-of-the-art baselines by detecting a significantly larger and more diverse set of violations. Crespo-Rodriguez et al. [17] proposed *PAFOT*, a position-based framework that models the movements of surrounding traffic agents using a nine-position grid around the ego vehicle and applies a genetic algorithm to evolve collision-inducing scenarios. To improve local search, *PAFOT* incorporates a local fuzzing strategy, which helps uncover additional safety violations near high-risk cases, leading to better performance than *AV-Fuzzer* and random testing in Carla. In a more recent study, Ji et al. [41] proposed *DoFuzz*, a search-based framework that generates naturalistic driving scenarios from traffic patterns respecting basic vehicle dynamics and trajectories learned from real driving data, and prioritizes test executions based on a freshness score that favors behaviorally novel cases. Evaluated on multiple Carla *Leaderboard* [11] driving tasks, *DoFuzz* discovered more unique failures and achieved faster convergence compared to *BehAVExplor*.

Algorithmic innovations have further enriched the field. Gambi et al. [28] integrated search-based testing with *Procedural Content Generation*, a technique for algorithmically creating diverse virtual environments, to automatically construct virtual road scenarios for testing lane-keeping functionality. Goss and Akbaş [30] proposed a two-phase search method that first explores the scenario space broadly and then refines around failure-inducing cases to focus testing on critical regions. Zheng et al. [91] proposed a quantum genetic algorithm to reduce the computational load associated with population size. Humeniuk et al. [37] introduced *AmbieGen*, a modular, search-based test-generation framework that uses evolutionary algorithms to identify fault-revealing scenarios for ADSs and robots. In a subsequent study, they proposed *RIGAA* [38], a reinforcement learning-informed genetic algorithm that integrates a pre-trained reinforcement learning agent into the evolutionary search process to improve the efficiency of simulator-based test generation. Evaluated on autonomous robot and vehicle systems, *RIGAA* significantly outperformed prior methods, including *AmbieGen*, revealing more failures in the vehicle domain and achieving faster convergence in both scenarios.

In summary, search-based testing has proven highly effective in uncovering critical failures in ASs by systematically exploring scenario spaces and guiding test generation toward high-risk behaviors and thereby surfacing edge cases. Techniques in this domain have evolved from early multi-objective formulations and surrogate-assisted search to more recent innovations involving fuzzing and behavioral clustering. Despite these advancements, search-based testing remains computationally expensive, particularly when dealing with high-dimensional scenario parameters and complex operating scenarios. This cost becomes a critical bottleneck in modern AS development pipelines, where systems are frequently retrained, fine-tuned, or updated to adapt to new data, hardware, or operating environments. These challenges highlight the need for scalable and adaptive testing frameworks that can efficiently identify critical behavioral changes across AS versions and minimize redundant simulation efforts, to keep pace with the rapid development cycles of modern ASs.

4.2 Metamorphic Testing of Autonomous Systems

Metamorphic Testing (MT) [12, 13] was introduced as a solution to the test oracle problem in domains where ground-truth output is hard to define. Recent work highlights the effectiveness of MT in many domains, including ASs, where it enables the detection of failures by checking the consistency of outputs across related inputs rather than relying on a known ground truth [71].

Lindvall et al. [52] developed a simulation-based testing framework that integrates MT with model-based testing to automatically generate and evaluate test cases for autonomous drones under variations of geometric and environmental conditions. Their approach revealed unstable behaviors and critical failure cases, including collisions and perception failures, demonstrating its effectiveness in uncovering subtle flaws that traditional methods often miss. Tian et al. [76] introduced *DeepTest*, which defines transformation-based MRs involving lighting and weather variations to identify behavioral inconsistencies in Deep Neural Network-based ADSs. This idea was expanded in *DeepRoad* [90], which applied generative adversarial networks to generate high-fidelity weather conditions, enabling more realistic and effective scenario transformations. Han and Zhou [31] employed a metamorphic fuzzing approach, using MRs to filter false positives by identifying genuine faults among automatically generated edge-case scenarios. In a more recent effort, Cheng et al. [15] developed *Decictor*, a scenario-based MT framework aimed at assessing optimal decision-making in ADSs. By applying minor, carefully controlled changes to the driving environment, *Decictor* identifies situations where the system diverges from the optimal trajectory without altering the ground-truth path. Fredericks et al. [26] proposed *DroneMR*, an MT framework for drone systems that combines evolutionary search with contextual MR inference to detect failures under uncertain conditions. Applied to a micro-drone platform in simulation, *DroneMR* generated diverse operating contexts and corresponding metamorphic test cases, showing its potential for both design-time and run-time validation in safety-critical settings.

Overall, MT provides a practical means to detect failures in both module-level and system-level testing of ASs, especially in the absence of a test oracle. Its power lies in the definition of MRs that express consistent behavioral expectations under controlled input modifications, enabling the detection of latent inconsistencies. Nonetheless, one major limitation lies in the difficulty of formulating comprehensive and safety-relevant MRs, especially in dynamic, real-world operating scenarios characterized by environmental variability and unpredictable agent interactions. The task of identifying effective MRs often requires significant domain expertise and manual effort, making automation a persistent open challenge [48]. However, because MRs encode general behavioral properties rather than exact output specifications, they can remain effective even when only partially specified, as violations of these MRs may still indicate anomalous or flawed behaviors.

4.3 Discussion

The current body of research identifies search-based testing and MT as the two principal methodologies for system-level evaluation of ASs. Search-based testing has proven effective at systematically exploring large, complex scenario spaces to uncover rare but safety-critical failures, benefiting from advancements such as surrogate modeling, behavior-guided fuzzing, and other innovative approaches. On the other hand, MT has emerged as a practical solution to the oracle problem, enabling validation in scenarios where ground-truth outputs are unavailable by checking the consistency of outputs under controlled input transformations defined by MRs.

Despite the progress achieved by existing techniques, several limitations remain that hinder their practical deployment in real-world AS development cycles. Most notably, current search-based approaches are computationally expensive, requiring extensive simulation time and resources to explore high-dimensional scenario spaces. This limitation becomes especially problematic in modern development settings, where ASs are continuously updated to accommodate new environments, system configurations, or operating policies. In such contexts, the cost of re-running full-scale test campaigns after each system update is prohibitive. Furthermore, current approaches assess ASs in isolation, so they cannot reveal whether a newly discovered failure is a longstanding flaw or a degradation introduced by the latest software update. This inability to distinguish degradations from persistent issues makes it difficult for developers to prioritize fixes, allocate debugging resources effectively, and assess the impact of recent updates on system safety. Finally, existing methods typically lack mechanisms for fault interpretability, offering little insight into why failures occur or which scenario attributes most contribute to inducing unsafe behaviors. This lack of explanatory power limits their utility in iterative development, where actionable feedback is essential for effective debugging and refinement.

Building on these observations, our work investigates how to integrate search-based testing and MT into a unified framework that addresses the practical limitations of current approaches. The central question we explore is how to design a testing methodology that remains effective and efficient in uncovering critical behavioral failures induced by system updates, while also providing insights into the key attributes of failure-inducing scenarios.

5 Our Approach

Fig. 1 presents an overview of our proposed approach, *Cooperative Co-evolutionary differential and Metamorphic Automated Generator for Interpretable Cases in autonomous systems (CoCoMagic)*. Our approach integrates a constrained archive-based CCEA with DT and MT to effectively and efficiently generate test cases that expose differences in MR violations between two versions of an AS. CCEA decomposes the original high-dimensional search problem into lower-dimensional subproblems, each handled by an independent population, not only enabling parallelism but also increasing the efficiency of the search process [56]. Specifically, we cast the problem as a two-population CCEA: the first population explores source scenarios, and the second consists of perturbations derived from the predefined MRs. Archive structures store elite individuals from both populations to guide the search toward promising regions of the space. Through collaboration, individuals from both populations combine to form complete test cases, i.e., complete solutions, that reveal differences in MR violations. To further guide the search, we use a set of MRs that share the same output relation, allowing the search to explore a broader, more diverse range of test cases while ensuring that each candidate test case directly targets relevant relation violations. Additionally, constraints ensure that the generated test cases remain as relevant and realistic as possible in real-world conditions. Finally, root cause analysis is performed on the generated complete solutions to interpret and explain the observed differences in MR violations between the two AS versions.

In the following sections, we first introduce a fitness evaluation framework (Section 5.1), which incorporates a joint fitness function [63, 87] for measuring the difference in MR violations in complete solutions formed through collaboration between individuals from the two populations, as well as an individual fitness function [63, 87] for assessing each individual's contribution. We then describe how constraints are incorporated into the search process to ensure that all generated solutions remain applicable to real-world conditions (Section 5.2). Next, we present our constrained CCEA-based search algorithm, which leverages the fitness functions and constraints to effectively guide the search (Section 5.3). Other components of the approach follow our previous approach

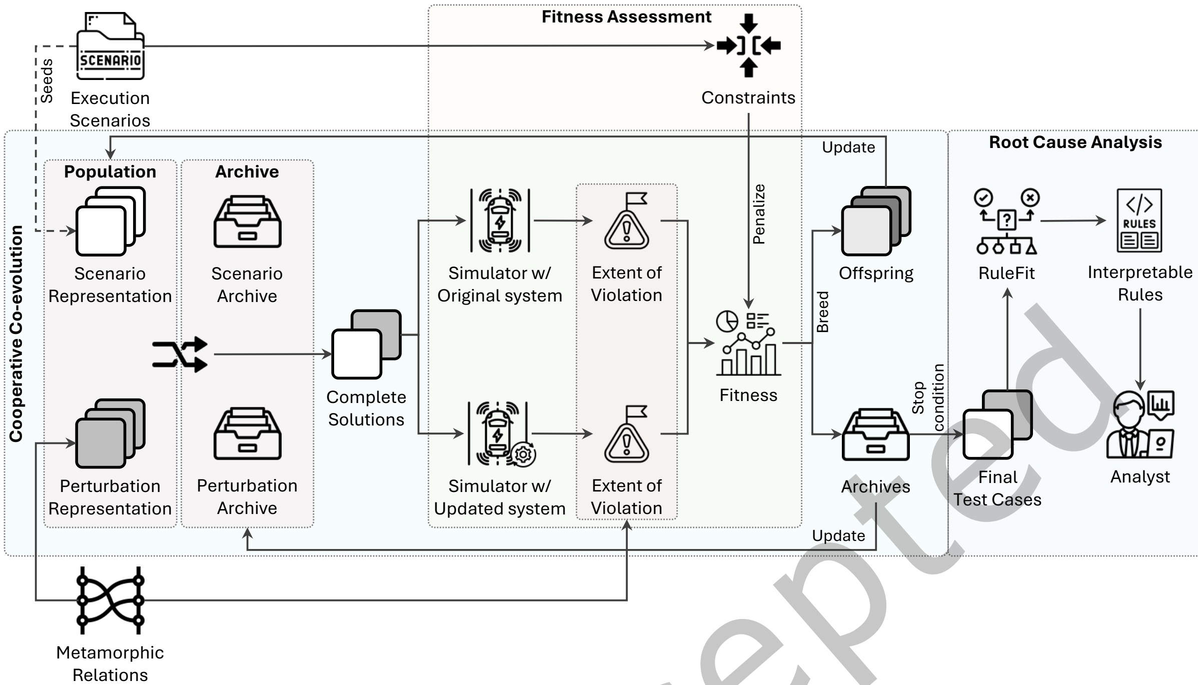

Fig. 1. Overview of *CoCoMagic*.

introduced in Section 2.5 and are reused here without modification. Finally, we describe the root cause analysis applied to the generated complete solutions, which helps interpret and explain the observed differences in MR violations between the two AS versions (Section 5.4).

5.1 Fitness Function

We aim to define fitness functions that effectively guide the search toward solutions that maximize the difference in MR violations between two versions of an AS. To this end, we first introduce a method for quantifying the extent of MR violation between a source and a follow-up scenario. We then extend this definition to compute the difference in MR violations between two versions of the AS.

Definition 2 (Extent of MR violation)**.** Given a system under test V, a source scenario s and a perturbation q derived from a set of MRs that share the same output relation *or*, and a function D_{or} which quantifies the deviation from *or*, we define the extent of violation of *or* as follows:

$$E(s, q, V, D_{or}) \coloneqq D_{or}(V(s), V(q(s))) \tag{1}$$

where $V(s)$ and $V(q(s))$ denote the outputs produced by the system V when executing the source scenario s and the follow-up scenario $q(s)$, respectively.

The exact formulation of D_{or} depends on the specific *or* of the MR in use, and thus the extent of violation is computed accordingly. For example, if *or* specifies that the output time series of the ego vehicle's speed should remain unchanged between the source and follow-up scenarios, D_{or} can be defined as the mean absolute difference between matched points in the two speed series. The matching can be performed either pairwise by time index or using more advanced alignment algorithms, e.g., to handle temporal shifts or unequal lengths. Accordingly, we can define D_{or} as follows:

$$D_{or}(T_s, T_q) := \frac{1}{|M|} \sum_{(t_s, t_q) \in M} (|t_s - t_q| - \phi)$$

where $M \subseteq T_s \times T_q$ denotes the set of matched point pairs between the two time series T_s and T_q, and ϕ is a tolerance threshold that accounts for acceptable deviations due to noise or minor variations.

Eq. (1) captures how much the behavior in the follow-up scenario deviates from what is expected. A higher value indicates a greater deviation, suggesting a more significant violation of the metamorphic relation. In Eq. (1), the pair (s, q) constitutes a complete solution.

Definition 3 (Difference in MR violations)**.** Given a source scenario s and a perturbation q derived from a set of MRs with the same output relation *or*, and two versions of an AS, denoted V_{ref} and V_{test}, the difference in MR violation between the two versions is defined as follows:

$$Diff(s, q) := |\max(E(s, q, V_{test}, D_{or}), 0) - \max(E(s, q, V_{ref}, D_{or}), 0)| \tag{2}$$

We use the maximum of E and 0 to ignore negative values of E, which indicate no violation of the given MRs. In such cases, the difference is not meaningful and should not affect fitness, as our primary objective is to identify complete solutions that reveal behavioral divergences in MR violations. Eq. (2) quantifies how differently the two system versions respond to the same set of MRs, highlighting changes that strengthen or alleviate MR violations. In our framework, *Diff* serves as the joint fitness function, which we aim to maximize during the search process to identify complete solutions that best expose behavioral discrepancies between the two versions.

To guide the optimization of each population independently, we define an individual-level fitness function that captures how well a given scenario or perturbation contributes to the discovery of differences in MR violations. This function assesses each individual based on its best-performing collaboration with members of the opposite population. For a source scenario, its fitness is computed as the highest violation difference it produces when paired with any perturbation from the current population. Conversely, for a perturbation, its fitness corresponds to the maximum violation difference observed across all source scenarios with which it is combined.

Definition 4 (Individual Fitness Function)**.** Formally, given an individual i, its fitness is defined as follows:

$$fitness(i) := \begin{cases} \max_j (Diff(i, q_j)) & \text{if } i \text{ is a scenario} \\ \max_j (Diff(s_j, i)) & \text{if } i \text{ is a perturbation} \end{cases} \tag{3}$$

where each q_j or s_j is drawn from the opposite population.

This approach encourages individuals to specialize in combinations that are most likely to reveal behavioral differences between system versions. Prior work has shown that relying on the maximum fitness obtained with collaborators, rather than the average or minimum, leads to more effective co-evolutionary outcomes [8, 9, 63, 80].

5.2 Constraints

During deployment, an AS continuously operates in an evolving environment and is therefore exposed to operating conditions that reflect the current state of the environment. When the system is updated (e.g., retrained) to adapt to such changes, these recently observed conditions become particularly important for evaluation, as they are up to date and grounded in real-world execution. To capture these conditions, we run simulations of the system under test, monitor environmental conditions as they arise, and collect a representative set of execution scenarios that characterizes operating conditions. This collected set is then used to constrain the subsequent search for the updated version, ensuring that generated test cases remain aligned with realistic execution-time scenarios to an extent configurable by engineers.

During the scenario collection phase, the framework continuously monitors system behavior and environmental conditions, encoding each encountered situation in terms of its salient features (e.g., environmental context and agent behavior). As new situations arise, their feature representations are compared against those of previously recorded scenarios, and only sufficiently distinct or representative scenarios are retained in order to avoid redundancy. Over time, this novelty-driven filtering process yields a compact yet diverse set of execution scenarios that captures a wide range of operational conditions and serves as a basis for constraining the search.

To enforce relevance, we quantify how similar a generated complete solution is to these real-world execution scenarios by measuring its dissimilarity to a set of execution scenarios $\mathbb{S}$ collected in execution time. This constraint biases the search toward solutions that are realistic and operationally grounded, ensuring that generated test cases reflect conditions the system has recently encountered and is likely to encounter again in practice.

Definition 5 (Dissimilarity to execution scenarios)**.** Given a source scenario s, a perturbation q derived from a set of MRs sharing the same output relation, and a set of execution scenarios $\mathbb{S}$, the dissimilarity of the complete solution formed by s and q to $\mathbb{S}$ is defined as:

$$Dissim(s, q, \mathbb{S}) \coloneqq \min_{i \in \{s, q(s)\}} (\min_{j \in \mathbb{S}} (dist(i, j))) \tag{4}$$

where $q(s)$ denotes the follow-up scenario and $dist(i, j)$ represents a distance measure between scenarios i and j.

A typical choice for *dist*, which accommodates both continuous and categorical variables, is the heterogeneous distance metric as introduced in Section 2.5.2. Eq. (4) measures how dissimilar a complete solution is from the set $\mathbb{S}$ by considering the minimum distance between either the source or follow-up scenario and its nearest neighbor in $\mathbb{S}$. This approach ensures that at least one component of each complete solution closely resembles a real execution scenario from $\mathbb{S}$, preserving relevance. Using the minimum rather than the average distance avoids inflating dissimilarity when a complete solution is very close to one scenario but far from others, which can result in a misleadingly moderate average that suggests the solution is farther from the set than it actually is. This constraint helps maintain the plausibility and practical relevance of the generated test cases, focusing the search on behavior differences that are not only theoretically interesting but also applicable in real-world AS deployments.

5.3 Algorithm

Building on the fitness functions defined in Section 5.1, the constraints described in Section 5.2, and other components reused from our previous approach introduced in Section 2.5, this section presents our proposed algorithm.

Algorithm 1 presents the pseudocode of our algorithm. It takes as input a population size n and a set of execution scenarios $\mathbb{S}$, and it outputs an archive X containing distinct complete solutions. The algorithm begins by initializing two populations: a scenario population P_s, which is either randomly generated or sampled from $\mathbb{S}$ (line 1), and a perturbation population P_q (line 2). Sampling the scenario population from execution scenarios enhances both the diversity of the initial population and its relevance to real-world conditions. The choice of initialization strategy influences the similarity of the resulting complete solutions to real-world scenarios, as further discussed in Section 7.3. Corresponding population archives X_s and X_q are created at lines 3 and 4. An empty archive of complete solutions X is also initialized at line 5. The core of the algorithm is a co-evolutionary process where P_s and P_q co-evolve, guided by X_s and X_q, until a predefined *stopping condition* is met (line 6). This process consists of the following iterative steps: (1) Evaluating the fitness of individuals in both P_s and P_q, updating the archive X with complete solutions and their joint fitness values as determined by the two system

Algorithm 1: Cooperative co-evolutionary algorithm.

Input: Population size n
Dissimilarity threshold θ
Execution scenarios $\$$
Output: Archive of complete solutions X

1 Population of scenarios $P_s \leftarrow$ *initPopulation*(n);
2 Population of perturbations $P_q \leftarrow$ *initPopulation*(n);
3 Archive of scenarios $X_s \leftarrow P_s$;
4 Archive of perturbations $X_q \leftarrow P_q$;
5 Archive of complete solutions $X \leftarrow \emptyset$;
6 **while** *not(stopping condition)* **do**
7 $\quad P_s, P_q, X \leftarrow$ *assessFitness*$(P_s, P_q, X_s, X_q, X, \theta, \$)$;
8 $\quad X_s \leftarrow$ *updatePopulationArchive*(P_s);
9 $\quad X_q \leftarrow$ *updatePopulationArchive*(P_q);
10 $\quad P_s \leftarrow$ *breed*$(P_s) \cup X_s$;
11 $\quad P_q \leftarrow$ *breed*$(P_q) \cup X_q$;
12 **end**
13 **return** X;

versions (line 7); (2) Updating the population archives X_s and X_q based on individual fitness values (lines 8–9); and (3) Breeding new individuals for P_s and P_q, and merging them with X_s and X_q to form the next generation of both populations (lines 10–11). After the *stopping condition* is satisfied, the algorithm returns the final archive X (line 13).

The function *assessFitness* initially computes joint fitness values for complete solutions formed by combining individuals from P_s, P_q, X_s, and X_q. To improve computational efficiency, complete solutions that already exist in X are excluded from re-simulation, avoiding redundant evaluations. After updating the archive X, each individual's fitness value is assigned based on the joint fitness obtained from the complete solutions it contributes to. Finally, fitness clearing [64] is applied to each population to maintain variety within the populations, reducing the risk of premature convergence [64] and the impact of genetic drift [69], which causes the population to converge towards individuals that resemble those with high fitness [24, 29, 36, 57].

The pseudocode for *assessFitness* is provided in Algorithm 2, with the following inputs: the population of scenarios P_s, the population of perturbations P_q, the population archive of scenarios X_s, the population archive of perturbations X_q, the archive of complete solutions X, the dissimilarity threshold θ, and the set of execution scenarios $\$$. The function returns updated populations P_s and P_q with assigned individual fitness values, along with an updated archive X incorporating newly created complete solutions and their corresponding fitness values. The algorithm begins by generating an initial set of complete solutions by combining individuals from both populations and population archives (line 1). Specifically, each individual in P_s collaborates with every individual in X_q, and vice versa for P_q and X_s. For each generated complete solution *cs* (line 2), if $cs \notin X$, i.e., *cs* has not been previously evaluated (line 3), it is evaluated by executing both versions of the AS (line 4). The difference in MR violations between the two versions is then quantified using Eq. (2) (line 5), and *cs*, along with its evaluated result, is added to X (line 6). Upon updating X, the algorithm assigns fitness values to all individuals in both populations. For each population $P \in \{P_s, P_q\}$ (line 9) and for each individual $i \in P$ (line 10), the individual fitness of i is determined as the maximum joint fitness obtained from all complete solutions involving i, calculated according to

Algorithm 2: assessFitness

Input: Population of scenarios P_s
Population of perturbations P_q
Archive of scenarios X_s
Archive of perturbations X_q
Archive of complete solutions X
Dissimilarity threshold θ
Execution scenarios $\mathbb{S}$

Output: Updated population of scenarios P_s
Updated population of perturbations P_q
Updated archive of complete solutions X

1 Set of complete solutions $CS \leftarrow collaborate(P_s, P_q, X_s, X_q)$;
2 **foreach** *Complete solution* $cs \in CS$ **do**
3   **if** $cs \notin X$ **then**
4         $execute(cs)$;
5         $cs.fitness \leftarrow Diff(cs.s, cs.q)$;
6         $X \leftarrow X \cup \{cs\}$;
7   **end**
8 **end**
9 **foreach** *Population* $P \in \{P_s, P_q\}$ **do**
10   **foreach** *Individual* $i \in P$ **do**
11         $i.fitness \leftarrow assessIndividualFitness(i, X)$;
12   **if** $Dissim(i.s, i.q, \mathbb{S}) > \theta$ **then**
13             $i.fitness \leftarrow \frac{i.fitness}{e^{c(Dissim(i.s,i.q,\mathbb{S})-\theta)}}$;
14   **end**
15   **end**
16     $fitnessClearing(P)$;
17 **end**
18 **return** P_s, P_q, X;

Eq. (3) (line 11). This strategy of assessing individual fitness using an elitist approach (i.e., selecting the individual with the highest fitness value) aligns with findings from previous experimental research [54, 56, 89]. Next, the algorithm applies a penalty to the individual fitness if the dissimilarity between the individual i and the execution scenario set $\mathbb{S}$, as defined in Eq. (4), exceeds the threshold θ (line 12–13). The penalized fitness is computed as:

$$fitness := \frac{fitness_{raw}}{e^{c(Dissim(i.s,i.q,\mathbb{S})-\theta)}} \tag{5}$$

where $fitness_{raw}$ denotes the individual fitness value before penalization and c is a constant that controls penalty severity. This exponential-decay penalty function [58] ensures that individuals with greater dissimilarity to execution scenarios receive fitness values that decrease exponentially. Finally, the algorithm performs fitness clearing on each population P (line 16). The algorithm concludes by returning the updated P_s, P_q, and X (line 18).

The function *updatePopulationArchive* updates the population archives X_s and X_q, which play a key role in guiding the search process. Each individual collaborates with archive members to form complete solutions, which are evaluated for fitness. Several strategies can be employed to update these archives, such as selecting individuals with the highest fitness values or choosing individuals at random [56]. To balance exploitation and exploration, a hybrid approach may be used, i.e., selecting the top-performing individual along with randomly chosen individuals [56, 74]. In our case study, we adopt a hybrid approach that selects the individual with the highest fitness and fills the remaining with individuals that maximize the overall diversity of the archive, as described in Section 2.5.2. This ensures that high-fitness solutions are retained while preserving diversity within the archive, promoting broader exploration of the search space, and helping to prevent premature convergence on suboptimal solutions.

The function *breed* generates new individuals by combining selected parents from the current populations. This process drives exploration of the search space and supports the discovery of better solutions over successive generations. It relies on selection, crossover, and mutation operators to produce offspring that inherit characteristics from their parents while introducing variation to maintain diversity. Our algorithm is designed to be flexible, allowing users to define and configure selection, crossover, and mutation operators tailored to the specific representations of scenarios and perturbations. This user-defined setup ensures adaptability to various ASs and problem contexts, while preserving compatibility with the structure and constraints of the search space. In the domain of ADS testing, the selection, crossover, and mutation operators described in Section 2.5.3 can be applied to both scenarios and perturbations.

5.4 Root Cause Analysis

After generating MR-violating solutions that elicit behavioral differences between two ASs, we introduce an interpretability approach designed to analyze the root causes of these differences in MR violations. Our goal is to uncover the underlying reasons behind the observed behavioral differences, enabling engineers to understand under which conditions the two versions behave differently and how significantly those conditions contribute to the differences. To this end, we extract interpretable rules that describe the behavioral differences between the two versions of an AS. These rules specify, in a human-readable format, the conditions under which differences occur and quantify the extent to which each rule contributes to the observed differences in MR violations. This analysis helps identify potential degradations in the updated AS and enables targeted improvements to enhance system safety, especially by focusing on high-impact conditions.

To extract interpretable rules that explain behavioral differences between two versions of an AS, we employ *RuleFit* [27], a method that generates human-readable rules derived from decision trees trained on input data. According to Margot et al. [60], *RuleFit* is the most accurate rule-based algorithm in their evaluation of predictive performance. *RuleFit* operates by first training an ensemble of decision trees (we use GBDTs in *CoCoMagic*) and converts each tree's decision paths into binary rules, where each rule represents a specific combination of feature conditions (e.g., ego vehicle speed > $70km/h$ and weather = foggy). These derived binary rules are then combined with the original features in a sparse linear regression model using Lasso regularization [77]. Lasso assigns weights to both raw features and derived rules, automatically filtering out less important ones through coefficient shrinkage, thereby improving interpretability and focusing attention on the most relevant patterns.

In our context, we apply *RuleFit* to analyze the differences in MR violations between two versions of an AS by training a model on the generated complete solutions. Each input vector represents a scenario-perturbation pair, i.e., a complete solution incorporating features from both the source scenario and the follow-up scenario resulting from applying perturbations. These features typically include continuous and categorical variables, such as ego vehicle initial speed, object positions, and environmental conditions relevant to testing ADSs. To ensure compatibility with *RuleFit*, preprocessing such as one-hot encoding is often required for categorical features, converting them into binary indicators that the model can process alongside continuous variables. The target variable is the observed difference in MR violations between the two system versions for each complete solution. In the end, the *RuleFit* model outputs a set of human-readable IF-clause rules along with coefficients and support values for the original and derived features. Each rule's coefficient quantifies its contribution to predicting the difference in MR violations, while the support value indicates the proportion of complete solutions where the rule's conditions hold.

By examining these rules, we can identify scenario patterns that most strongly influence behavioral differences between the two versions. For example, a rule such as "IF ego vehicle speed > $70km/h$ AND foggy weather = true, THEN the difference in MR violation increases by 2.5" suggests that under high-speed, foggy conditions, the two system versions behave more inconsistently, with the updated system likely exhibiting a higher MR violation. The increased value of 2.5 indicates the magnitude of this effect on the difference in MR violations. If the average difference in MR violations across all complete solutions is, for instance, 1.0, this rule highlights a significant deviation from the norm. Such insights help developers and testers focus their efforts by highlighting specific scenario conditions where updates may have introduced unintended behaviors or safety risks. These interpretable rules directly support both root cause analysis and actionable decision-making, such as prioritizing which scenario types to investigate further or monitor more closely in future system updates.

6 Empirical Evaluation

In this section, we empirically evaluate the effectiveness and efficiency of *CoCoMagic* for generating test cases that reveal differences in MR violations between two versions of the AS under test, comparing it against two baseline methods. We further investigate how the use of constraints and population initialization strategies affects the similarity between the identified test cases and the execution scenarios that represent real-world operating conditions. Finally, we assess how our interpretability approach helps capture the underlying root causes of the observed differences in MR violations between the two ASs under test. Specifically, we answer the following Research Questions (RQs).

RQ1: How effectively can *CoCoMagic* find test cases revealing differences in MR violations compared to baseline methods?

To answer RQ1, we evaluate the performance of different search methods in terms of the number of distinct complete solutions identified that reveal the differences in violations of predefined MRs within a given search budget.

RQ2: How efficiently can *CoCoMagic* find test cases revealing differences in MR violations compared to baseline methods?

To answer RQ2, we examine the speed at which different search methods identify distinct complete solutions that reveal the differences in violations of predefined MRs.

RQ3: To what extent do the applied constraints and population initialization strategies affect the similarity of identified test cases to execution scenarios?

To answer RQ3, we analyze the impact of constraints and population initialization strategies. First, we evaluate how these factors affect the similarity between the identified complete solutions and the execution scenarios. Next, we assess their effect on the overall effectiveness of the search process. Understanding these aspects is essential to determine whether our approach effectively guides the search toward more realistic solutions and to identify any potential negative side effects.

RQ4: To what extent does our interpretability approach contribute to identifying the root causes of the differences in MR violations?

To answer RQ4, we examine the accuracy and support of interpretable rules, extracted from the identified complete solutions, in explaining the observed differences in MR violations. Further, we ask four industrial experts to provide feedback on the rules via a questionnaire study.

6.1 Subject Systems and Simulation Environment

Our experiments were conducted using CARLA [22], a widely used open-source simulator tailored for autonomous driving research. CARLA enables the creation of complex, interactive driving scenarios featuring configurable weather, traffic, and environmental elements. To ensure realistic perception and behavior, we incorporated INTERFUSER [73], a state-of-the-art multi-modal end-to-end ADS that ranks among the top performers on the official CARLA *Leaderboard* [11]. INTERFUSER fuses data from LiDAR and Camera sensors to deliver rich scene information.

To compare two realistic iterations of the same ADS, we considered two versions of INTERFUSER, namely a reference version (V_{ref}) and an updated test version (V_{test}). V_{ref} corresponds to the original INTERFUSER model released by its authors. To obtain V_{test}, we followed the official retraining workflow provided with the framework. Specifically, we generated an additional training dataset using the provided data-collection pipeline. We then retrained the original INTERFUSER model on this newly collected dataset, thereby yielding an updated version. This construction of V_{ref} and V_{test} mirrors a common iteration cycle in practical AS development, where an existing model or component is updated by retraining on newly acquired operational data.

Experiments were executed on a dedicated high-performance server equipped with 56 CPU cores and 2 GPUs (each with 48GB of memory). To increase throughput, we deployed 4 parallel CARLA instances in separate Docker containers, leveraging CUDA 12.4 for GPU-accelerated computation. This setup enabled concurrent scenario evaluations at scale using CARLA version 0.9.10.1.

6.2 Baseline Methods

Since our work is the first to systematically generate test cases aimed at revealing differences in MR violations between two versions of an AS, there are no existing methods tailored for this exact problem. To enable meaningful comparison, we evaluate *CoCoMagic* against two baseline strategies that reflect commonly used search paradigms in test generation.

- ***Random Search (RS):*** This baseline involves generating test cases through uniform random sampling without any guidance from fitness feedback. Although simple, *RS* provides a useful reference point to assess the inherent difficulty of the search and the value of guided optimization.
- ***Standard Genetic Algorithm (SGA):*** This variant applies a standard genetic algorithm [35] using a single population, treating the entire solution space as a uniform search domain. Unlike our method, it does not differentiate between scenario structure and perturbation behaviors. Comparing with *SGA* helps highlight the benefits of our cooperative co-evolutionary setup.

By including these two baselines, we aim to illustrate both the relative performance gains achieved through guided search and the contribution of our modular co-evolutionary design to the quality and diversity of the generated test cases.

To investigate the impact of constraining mechanisms and population initialization strategies on the effectiveness of *CoCoMagic*, we designed different variants of *CoCoMagic* with specific configurations. The first configuration represents the standard constrained version, where the search is forced to remain within a specified similarity to recorded execution scenarios. In the second configuration, denoted as *CoCoMagic\c*, the constraint was removed to assess its influence on the quantity and quality of generated test cases. To evaluate the effect of the initialization strategy, we introduced an alternative method, termed *Runtime Initialization (RI)*, in which the initial population is sampled directly from previously collected execution scenarios rather than generated randomly. We applied this initialization strategy to both the constrained and unconstrained variants, yielding 4 experimental configurations in total.

To assess the interpretability of the rules extracted by our approach, we compare *RuleFit* used in *CoCoMagic* against two baseline methods, i.e., *Random Forest (RF)* and *Gradient Boosted Decision Tree (GBDT)*, both commonly used for regression tasks.

6.3 Execution Scenarios

We collected a diverse set of execution scenarios by running InterFuser in Carla *Leaderboard* environments. Carla *Leaderboard* is a standardized evaluation framework that enables running an ADS through a variety of driving tasks in simulated urban environments. We collected these scenarios across four towns and eight distinct weather conditions, including clear skies, light rain, and heavy rain, under varying lighting conditions, e.g., noon and sunset.

To ensure temporal and spatial diversity, we sampled execution scenarios at fixed time intervals (every 5 seconds), with a random offset from the start of each route execution. We also limited the number of sampled scenarios per route-weather configuration to encourage balanced coverage, resulting in approximately 800 distinct execution scenarios used for our experiments. These execution scenarios are used for the constraining mechanism and population initialization strategy in *CoCoMagic* and its variants.

6.4 Metamorphic Relations

Table 1. MRs involved in the experiments.

Category	**MR**	**Description**
GP_1 Speed Reduction	MR_1^{*}	If a pedestrian appears on the roadside, then the ego vehicle should slow down.
	$MR_2^{*,\dagger}$	If the driving time changes into night, then the ego vehicle should slow down.
	$MR_3^{\dagger}$	Adding a vehicle in the front of the ego vehicle, the speed of the ego vehicle should decrease.
	$MR_4^{\dagger}$	Adding a pedestrian in the front of the ego vehicle, the speed of the ego vehicle should decrease.
	$MR_5^{\dagger}$	Changing from sunny to rainy, the speed of the ego vehicle should decrease.
GP_2 Actor Invariance	$MR_6^{\ddagger}$	In a given driving scenario where the ego vehicle detects a target obstacle and attempts to avoid a collision, the ego vehicle should keep the steering unchanged when the speed of the ego vehicle changes (increased or decreased by a certain factor).
	$MR_7^{\ddagger}$	In a given driving scenario where the ego vehicle detects a target obstacle and attempts to avoid a collision, the ego vehicle should keep the steering unchanged when adjusting (i.e., scaled down or up) the size of the ego vehicle.
	$MR_8^{\ddagger}$	In a given driving scenario where the ego vehicle detects a target obstacle and attempts to avoid a collision, the ego vehicle should keep the steering unchanged when adjusting (i.e., scaled down or up) the size of the target obstacle.
	$MR_9^{\ddagger}$	In a given driving scenario where the ego vehicle detects a target obstacle and attempts to avoid a collision, the ego vehicle should keep the steering unchanged when changing the position of the ego vehicle.
	$MR_{10}^{\ddagger}$	In a given driving scenario where the ego vehicle detects a target obstacle and attempts to avoid a collision, the ego vehicle should keep the steering unchanged when changing the speed of the target obstacle.
	$MR_{11}^{\ddagger}$	In a given driving scenario where the ego vehicle detects a target obstacle and attempts to avoid a collision, the ego vehicle should keep the steering unchanged when adding additional actors.

* These MRs are derived from study [20].
† These MRs are derived from study [19].
‡ These MRs are derived from study [40].

We used a set of 11 MRs from Table 1, drawn from established research [19, 20, 40], selected for compatibility with both the ADS under test and the simulation platform. Some MRs from prior literature were excluded due to incompatibility with the constraints of our simulator. For instance, an MR that requires replacing background buildings with trees could not be used, as it would require modifying the static map provided by CARLA, which is not feasible within our experimental setup. When applying these MRs, we ensured that the generated source scenarios satisfied their preconditions. For example, to apply MR_6, the source scenario must contain a target obstacle that the ego vehicle can detect and attempt to avoid. Consequently, the selected MRs are both valid and implementable within the CARLA simulation environment. Although these MRs are specified at a high level of abstraction, they can be concretely instantiated through appropriate parameterization and scenario configuration. For instance, changing the position of the ego vehicle can be realized by adjusting its spawn point from one lane to another in the CARLA simulator.

To facilitate evaluation, we grouped the MRs into two categories, i.e., GP_1 and GP_2, based on their output behavior. GP_1 includes MRs with expected speed reductions in response to environmental or traffic changes. GP_2 captures invariance-based relations, i.e., changes in actors should not significantly alter the vehicle's steering behavior.

In the following subsections, we report the results for GP_1. The outcomes for GP_2 are consistent with those of GP_1, thereby demonstrating the robustness of *CoCoMagic* across different MR groups. To avoid redundancy, the detailed results for GP_2, including all supporting figures, are presented in the appendix.

6.5 Parameter Settings

All methods were executed with a simulation budget of 500 scenario evaluations, chosen based on pilot experiments that demonstrated sufficient convergence behavior under this limit. To ensure fair comparisons, each method was allowed to use up to 500 scenario evaluations. Additionally, to mitigate the impact of simulator nondeterminism, we applied a re-evaluation procedure in the simulator-based violation assessment. Each test case was first simulated once, and if its resulting MR violation exceeded a minimum threshold of 0.2, it was re-executed twice. The final violation value was then obtained by aggregating the three violation values using their median. This strategy was adopted to limit the effect of simulation noise on promising candidates while keeping the computational cost manageable.

We configured each method to use a population of 5 individuals, balancing between computational cost and search effectiveness. Given the high cost of each fitness evaluation (approximately two minutes per test case), this smaller population size aligns with common practice in search-based testing literature [3, 16]. For both *CoCoMagic* and *SGA*, we adopted tournament selection with a size of 3, which strikes a good balance between exploration and exploitation [42, 46, 79]. This choice also helps avoid premature convergence, especially in small populations. The *SGA* baseline employs the same fitness function as the joint fitness function used in *CoCoMagic*, but applied to a single population whose individuals have the same structure, consisting of both source scenarios and perturbations. Accordingly, we employed the same mutation and crossover operators for both *CoCoMagic* and *SGA*, with the mutation and crossover rates set to 0.2 and 0.8, respectively, consistent with recommendations in the evolutionary computation literature [62, 72]. As for *CoCoMagic*, which relies on cooperative co-evolution, we tuned its specific hyperparameters via lightweight exploratory runs. The final settings included an archive size of 3 per population. The penalty severity constant c in Eq. (5) was set to 6.66 based on pilot experiments that balanced constraint satisfaction and search exploration.

Due to the nature of population-based algorithms, some methods may slightly exceed the 500-simulation budget per run, since evaluations occur in batches. In such cases, the actual number of simulations consumed per generation can vary between methods, leading to misalignment in budget usage. To ensure consistent comparisons, we applied linear interpolation on all evaluation metrics at points where the actual simulations deviated from the predefined budget. This process estimates metric values precisely at the intended search budgets by interpolating between the closest available points, enabling a fair assessment under an aligned computational budget.

6.6 Evaluation Metrics

To evaluate the effectiveness of different methods in identifying test cases that reveal differences in MR violations between two ADSs, we introduce the following metrics:

- ***Distinct Solutions (DS)*** denotes the number of distinct complete solutions that expose significant behavioral differences between the ADS versions. We require such solutions to satisfy the following conditions: (1) its fitness value, i.e., the measured difference in MR violations across ADS versions, exceeds a predefined fitness threshold θ_f, and (2) it is sufficiently different from other selected solutions based on a minimum distance threshold θ_d. Formally, let CS_V be the set of complete solutions identified by method V that satisfy the above criteria. The number of distinct solutions is defined as $DS(V) := |CS_V|$.
- ***Fitness*** quantifies the degree to which a complete solution exposes behavioral differences in MR violations between the two ADSs, with larger absolute values indicating more pronounced differences.

To evaluate the efficiency of different methods, we analyze the *DS* metric as a function of the search budget and employ the following additional metrics:

- ***Area Under the Curve (AUC)*** measures the area under the *DS*-budget curve, providing an aggregate measure of how quickly a method accumulates distinct complete solutions over the course of the search, where higher values indicate that a method finds more distinct solutions earlier during the search. This metric is particularly suitable for our setting, as it captures both the speed and volume of distinct solutions discovered, two central factors in assessing test generation efficiency. It allows us to compare algorithms not only at final budget levels but throughout the entire simulation horizon.
- ***Execution Time*** records the wall-clock time taken by each method to complete the search within the allocated budget. This metric provides complementary insights into the algorithms' computational efficiency, highlighting differences in their computational costs and offering a practical perspective on their suitability for cost-sensitive testing scenarios.

To evaluate the realism of the generated test cases, we employ the following metric:

- ***Average Execution Distance (AED)*** measures how closely the generated complete solutions resemble real-world execution scenarios. For each test case, we compute its distance to the nearest execution scenario (see Eq. (4)) and then aggregate these values by averaging over all test cases in a given run. This metric provides insights into how well the generated scenarios align with observed real-world scenarios.

To evaluate the interpretability of the rules extracted from the identified complete solutions, we utilize the following metrics:

- ***Mean Absolute Error (MAE)*** is a widely used metric for evaluating the accuracy of predictive models, measuring the average magnitude of errors between predicted and actual values [39, 82]. In our context, *MAE* quantifies how accurately the interpretable rules characterize the differences in MR violations between two versions of the ADS under test. A lower *MAE* value indicates greater model accuracy in capturing behavioral differences. Specifically, *MAE* calculates the mean of the absolute differences between each pair of predicted and true values:

$$MAE := \frac{1}{|\mathcal{SP}'|} \sum_{i \in \mathcal{SP}'} |y_i - \hat{y}_i|$$

where $\mathcal{SP}'$ denotes the set of complete solutions, y_i is the observed difference in MR violations for a given complete solution, and $\hat{y}_i$ is the predicted value.

- ***Support*** evaluates the extent to which the extracted interpretable rules cover the identified complete solutions, which is essential for their practical usefulness. A complete solution is considered supported if there exists at least one rule whose conditions are satisfied by the solution. A higher support indicates that the rules effectively capture the characteristics of a larger portion of the identified complete solutions.

6.7 Experiment Design

6.7.1 Design for RQ1. To answer RQ1, we evaluate the effectiveness of *CoCoMagic* in identifying test cases, i.e., complete solutions, that amplify safety-related behavioral differences between two versions of InterFuser. Unlike existing MR approaches that target outright violations of predefined properties, our objective is to identify solutions that exhibit large discrepancies in their effects on the two ADS versions. We compare *CoCoMagic* against the two baseline methods under the same search budget, using the *DS* metric as the primary indicator of effectiveness.

To ensure a comprehensive evaluation, we assess each method under multiple configurations of the fitness threshold θ_f and distance threshold θ_d. The fitness threshold controls the required degree of behavioral difference between the two ADSs, reflecting the severity of the identified complete solutions; higher values correspond to greater differences. The distance threshold, on the other hand, governs the minimal dissimilarity between retained solutions, thus controlling the diversity of the complete solutions. By exploring a wide range of threshold values, we can compare the performance of different approaches under varying demands for the severity and diversity of the generated complete solutions. Specifically, we use $\theta_f \in \{1.0, 1.3, 1.5, 1.8, 2.0, 2.3\}$ to cover a meaningful spectrum of severity constraints applied on the generated test cases. This range was chosen to approximately span the 80th percentile of the fitness values observed in our experiments (around 2.3), ensuring that the selected thresholds capture realistic and informative levels of severity. Solutions with fitness below 1.0 are excluded due to their limited discriminative value and minimal impact. For θ_d, we consider values from 0.0 to 4.0 in increments of 0.4, covering a range of diversity requirements from lenient to stringent. The upper bound was set to the 95th percentile of pairwise distances across all generated solutions, ensuring the threshold range meaningfully reflects the observed distribution of solution diversity.

To complement the *DS* metric, we also assess the quality of solutions generated by each method based on their fitness values. Specifically, we analyze the distribution of fitness values across all complete solutions produced by each approach, where a higher fitness indicates greater differences in MR violations. This provides insight into each method's capacity to uncover critical test cases.

6.7.2 Design for RQ2. To address RQ2, we analyze the rate at which different methods discover complete solutions over the course of their search. To this end, we track the number of distinct solutions identified by each method as a function of the search budget, expressed as a percentage of the total number of simulations allowed. The methodology follows that of RQ1, employing the same hyperparameters and repeating each method 10 times.

In addition to analyzing *DS* as a function of the search budget, we compute the *AUC* of each *DS*-budget curve and measure the execution time for each method, providing a comprehensive assessment of their efficiency in generating distinct complete solutions.

6.7.3 Design for RQ3. To address RQ3, we analyze how different configurations of *CoCoMagic* affect both the similarity of the identified test cases to execution scenarios and the overall effectiveness of the search process. Each variant was executed independently 10 times to account for stochastic variation in the search process. For all runs, we employed three metrics: (1) the *DS* metric, capturing the number of generated distinct solutions, (2) the average fitness of the test cases found by each variant, quantifying the capacity of uncovering critical test cases with severe behavioral discrepancies, and (3) the *AED* metric, measuring how closely the generated test cases resemble real-world execution scenarios.

6.7.4 Design for RQ4. To answer RQ4, we begin by assessing the accuracy of the interpretable rules derived from the complete solutions by evaluating the *MAE* metric. *RuleFit*, used in *CoCoMagic*, as well as the two baseline methods, *RF* and *GBDT*, are trained on the same dataset of complete solutions paired with their corresponding differences in MR violations. This dataset combines results from all search methods used in RQ1 and RQ3 and contains two types of complete solutions. The first type includes solutions where the two systems exhibit observable behavioral differences. Each of these solutions is assigned a fitness value that reflects both the direction and severity of the difference, with positive values indicating better performance by the original system and negative values indicating better performance by the updated system. The second type consists of solutions where no behavioral difference is observed between the systems. These solutions are assigned a fitness value of zero. This combination allows the prediction methods to learn not only what conditions lead to discrepancies, but also which solutions result in consistent behavior across both versions. By training on this combined dataset, *RuleFit* and baseline methods can identify the conditions that cause the systems to diverge, while also recognizing when their behaviors remain the same. The combined dataset consists of 2,573 complete solutions, with 1,967 solutions exhibiting behavioral differences and 606 solutions showing no differences, split into 80% for training and 20% for validation. To account for randomness during training, each method is evaluated across 500 independent runs with different random seeds to ensure robust and generalizable results. To facilitate interpretability, *RuleFit* is constrained to extract at most 30 rules per run. Since our goal is explanation rather than maximal predictive accuracy, a smaller, high-importance rule set is easier to audit and apply in practice. This threshold was chosen based on both empirical observations and considerations of human interpretability. In preliminary experiments, we varied the maximum number of rules while keeping the dataset of complete solutions fixed and evaluated the resulting prediction accuracy. The model achieved its highest accuracy when the limit was set to approximately 30 rules. From a cognitive perspective, prior work on model interpretability suggests that human analysts can effectively process only a limited number of rules before comprehension and decision quality degrade [45]. Therefore, limiting the output to 30 rules preserves most of the explanatory power while keeping the results concise and practical for human review. For a fair comparison, the hyperparameters of the baseline methods, including the maximum tree depth, the maximum number of leaf nodes, the number of trees, and other relevant parameters, were tuned using randomized search [5], a hyperparameter optimization technique that samples parameter configurations from predefined ranges, aiming to approximate the best-performing settings for each method on the dataset.

To understand how thoroughly the rules support the identified complete solutions with behavioral differences (1,967 solutions), we analyze the distribution of the rules' *support* across 500 independent runs of *RuleFit*. Specifically, in each run, we compute the proportion of complete solutions that are supported by the full set of extracted rules. This analysis reveals the extent to which the rules capture the behavioral discrepancies reflected in the identified complete solutions. A higher proportion indicates stronger support or broader coverage, suggesting that the rules provide a more comprehensive explanation of the behavioral differences between the two system versions.

To complement the quantitative evaluation of the interpretability framework, we conducted a questionnaire study with four industrial domain experts to evaluate the practical value of the generated rules. The four participants are professionals in the autonomous driving domain with experience ranging from 2 to 9 years. Their expertise primarily covers perception, decision-making, simulation and testing, and safety-related aspects of autonomous systems. All participants have prior involvement in safety-critical systems or risk assessment, most with extensive experience. Informed consent was obtained from all participants prior to the study for the use of their feedback. Fig. 2 presents a sample item from the questionnaire presented to the experts in this study. The full questionnaire is available in the replication package.

Each expert was provided with the top 10 interpretable rules, ranked by importance, that characterize the behavioral discrepancies between two ADS versions. The rules were first extracted from raw *RuleFit* output and translated into natural-language descriptions that are easier for practitioners to understand. This translation was performed through a systematic post-processing step described below.

Each rule consists of one or more conditions connected by logical conjunctions. To interpret a rule, we first analyzed each condition by decomposing the corresponding feature name into its semantic components. For example, the feature named *follow_up_walker_left_yaw_max_sin* is decomposed into: follow-up scenario, pedestrian, front-left region, heading direction, maximum value, and sine component. Accordingly, this feature represents the maximum sine value of the heading direction of pedestrians located in the front-left region of the ego vehicle in the follow-up scenario.

Next, threshold values were interpreted in accordance with the attribute's meaning and range. For directional attributes such as *yaw* and *angle*, values were interpreted relative to the ego vehicle's heading using CARLA's left-hand coordinate system. For instance, the condition *follow_up_walker_left_yaw_max_sin* > 0.4 indicates that, in the follow-up scenario, at least one pedestrian ahead on the left of the ego vehicle has a heading direction with a positive sine value of significant magnitude. Since positive sine values indicate rightward movement, and negative values indicate leftward movement, this condition can be interpreted as indicating that at least one such pedestrian is moving rightward. For distance and speed attributes, threshold values are interpreted relative to the scaled range of each feature, where values closer to zero indicate closer proximity or lower speed, and higher values indicate greater distance or higher speed.

After each condition in a rule was interpreted, the resulting condition-level interpretations were combined to derive the overall meaning of the rule. Finally, the resulting rule interpretation was reformulated into a coherent scenario description through minor manual rephrasing to improve clarity and readability. This final description preserves the rule's essential content while presenting it in a form easier for engineers to read and understand.

Each translated rule was presented together with its expected relative effect, its occurrence frequency across the generated test cases, and an illustrative example scenario in which the rule applies. The experts were then asked to assess the understandability and actionability of each rule by answering two questions on a five-point Likert scale [51]: (Q1) "How easy is it to understand the conditions described by this rule?" (Rate from 1 = Very difficult to understand to 5 = Very easy to understand), and (Q2) "Does this rule help you narrow down which types of conditions to prioritize for testing the updated system?" (Rate from 1 = Not at all to 5 = Very much).

Finally, we present several example rules extracted from one of the *RuleFit* runs to illustrate the interpretability and practical value of the generated rules. It showcases the interpretability approach's effectiveness in gaining a deeper understanding of the underlying factors contributing to the observed behavioral discrepancies.

7 Results and Analysis

In this section, we present the results and analysis of our empirical evaluation of *CoCoMagic*.

7.1 RQ1: Effectiveness of CoCoMagic

Fig. 3 reports the average *DS* values across 10 distinct experiments, achieved by *CoCoMagic*, *SGA*, and *RS*, at varying θ_d and θ_f. Each subplot shows the average *DS* values over executions, as a function of θ_d, with 95% confidence intervals represented by error bars. Different subplots correspond to different fitness thresholds θ_f.

The performance advantage of *CoCoMagic* is most evident at low and moderate distance thresholds. For every fitness threshold θ_f, *CoCoMagic* achieves the highest average *DS* values when θ_d is relatively small, often by a

Conditions Under Which the Two ADS Versions Differ

Vehicles to the left of the ego vehicle are traveling at fairly consistent speeds. Static obstacles ahead are directly in front of or to the right of the ego vehicle.

Observed Behavioral Discrepancy

On average, the updated system performs moderately better under these conditions than the original system.

Occurrence Frequency

This rule applies to 5.74% of all detected behavioral discrepancies.

An Example Scenario Where This Rule Applies

Side-by-side recordings of the two ADS versions executing the corresponding follow-up scenario.

Questions

Q1. How easy is it to understand the conditions described by this rule?

1 = Very difficult
2 = Difficult
3 = Moderately easy
4 = Easy
5 = Very easy

Q2. Does this rule help you narrow down which types of conditions to prioritize for testing the updated system?

1 = Not at all
2 = Slightly
3 = Moderately
4 = Considerably
5 = Very much

Fig. 2. Example of one questionnaire item used in the expert study. For each rule, experts were shown a natural-language description of the conditions, the observed behavioral discrepancy, the occurrence frequency, an illustrative example scenario, and the Likert-scale questions used for evaluation.

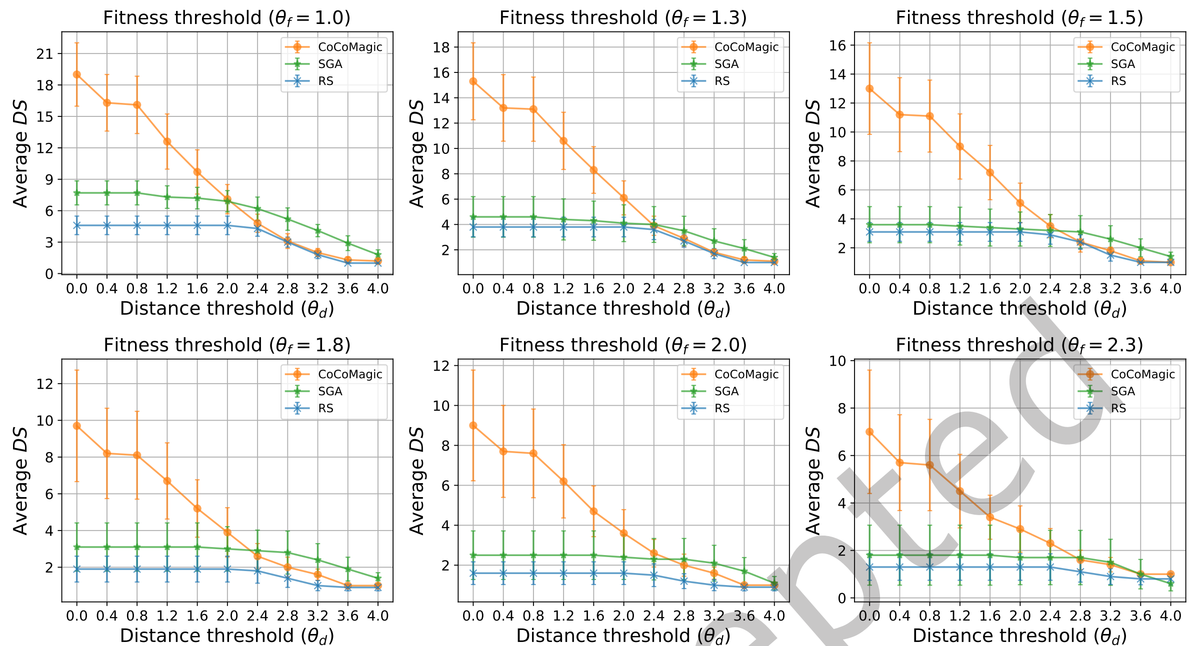

Fig. 3. *Distinct Solutions (DS)* vs. distance threshold (θ_d) across different fitness thresholds (θ_f). A higher *DS* value indicates more distinct test cases discovered by the method. Each curve plots the average *DS* value at different θ_d settings, under a specific fitness threshold θ_f. This figure reveals how each method balances the quantity and diversity of test cases as θ_f varies.

large margin over both *SGA* and *RS*. This trend remains consistent as θ_f is increased, indicating that *CoCoMagic* is substantially more effective at discovering a larger set of complete solutions across all severity constraints.

As θ_d increases, the average *DS* value decreases for all methods, which is expected since stricter diversity requirements lead to the exclusion of more solutions, and thereby reduce the number of retained complete solutions. However, the rate and implications of this decrease differ across methods. *CoCoMagic* shows a sharper decline as it begins with a much larger pool of archived solutions, and progressively fewer of them satisfy increasingly strict distinctness requirements. In contrast, *SGA* and *RS* remain relatively flat over a wider range of θ_d. Specifically, *SGA* begins to outperform *CoCoMagic* under strict diversity requirements and relatively permissive fitness thresholds ($\theta_d \geq 2.4$ and $\theta_f \leq 1.8$). This indicates that *SGA* may produce a solution set with slightly greater diversity, which only becomes advantageous under sufficiently strict distance thresholds. In other words, the main advantage of *CoCoMagic* lies in the low-to-moderate θ_d region, where it consistently produces more archived distinct solutions, whereas *SGA* becomes competitive only after aggressive diversity filtering removes much of that advantage. *RS* remains generally inferior throughout, and only approaches the other methods when all techniques yield very few remaining solutions.

Beyond the *DS* metric, we evaluate the quality of complete solutions produced by each method, excluding those with insignificant fitness values (< 0.25). This thresholding serves as a post-processing step to filter out solutions with near-zero fitness, which typically arise from randomness in the process and do not reflect genuine differences in violations. Fig. 4 presents the distribution of average fitness of complete solutions generated by *CoCoMagic*, *SGA*, and *RS* across 10 independent executions. *CoCoMagic* achieves both higher median and mean

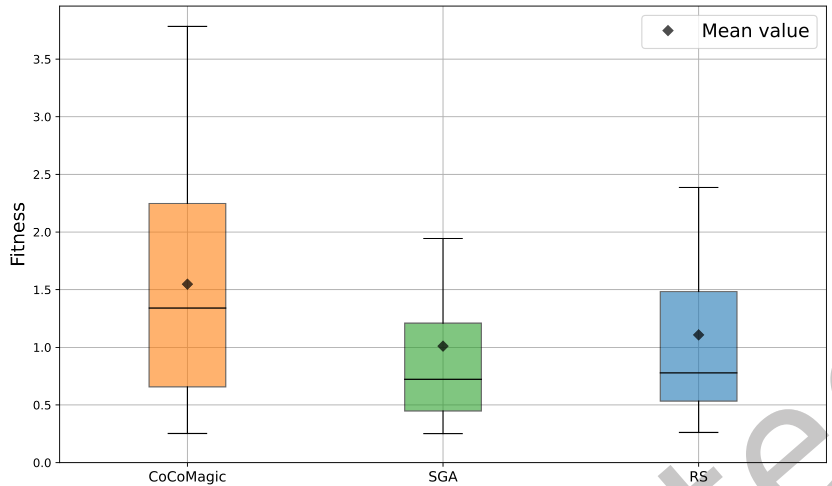

Fig. 4. Fitness distribution of test cases generated by *CoCoMagic*, *SGA*, and *RS*. Higher fitness indicates that the discovered test cases induce greater differences in the violations exhibited by the two ADSs. Each box shows the fitness values for the test cases across 10 executions, providing insight into each method's peak effectiveness. This figure highlights the relative strength of each method in uncovering critical test cases with severe behavioral differences.

fitness values compared to the baselines, indicating a greater capacity to uncover solutions that induce stronger behavioral discrepancies between the two system versions. On average, solutions generated by *CoCoMagic* reach a fitness value of 1.55, representing a 53% (p-value $< 10^{-10}$) increase over *SGA* (1.01) and a 40% (p-value $< 10^{-4}$) increase over *RS* (1.11). To assess the statistical significance of these differences, we conducted non-parametric Mann-Whitney U tests with Fisher's method for p-value correction [59].

In addition to the higher median and mean fitness, Fig. 4 highlights that *CoCoMagic* also attains a substantially larger range of fitness values compared to the baselines. While *SGA* and *RS* peak at fitness levels around 1.95 and 2.35, respectively, *CoCoMagic* reaches up to 3.80. This indicates that *CoCoMagic* not only consistently yields stronger solutions on average but also uncovers more severe cases that trigger significantly larger behavioral differences between the two system versions.

To provide a concrete illustration of the types of discrepancies captured by *CoCoMagic*, consider the scenario depicted in Fig. 5. The source scenario demonstrates the ego vehicle approaching and traversing an intersection under clear weather conditions. The perturbation generated by *CoCoMagic* includes two controlled changes: (i) placing a pedestrian on the crosswalk ahead of the ego vehicle, and (ii) modifying the weather from clear to rainy. The associated MR specifies that, under these changes, the ego vehicle should exhibit more cautious behavior, particularly by reducing speed when approaching the pedestrian.

Both the reference (original) and test (retrained) versions of *InterFuser* are evaluated on this test case. The reference system responds as expected by slowing down and stopping at a safe distance from the pedestrian, thereby satisfying the MR. In contrast, the test system maintains its speed and passes the pedestrian at close proximity, failing to exhibit the required behavioral adjustment. This divergence results in a violation of the MR in the test version (with a violation extent of 1.14), but not in the reference version, indicating a degradation with respect to the specified MR.

In summary, the results clearly demonstrate the superior effectiveness of *CoCoMagic* compared to both baseline methods in identifying a greater number of distinct test cases that not only satisfy diversity requirements but also

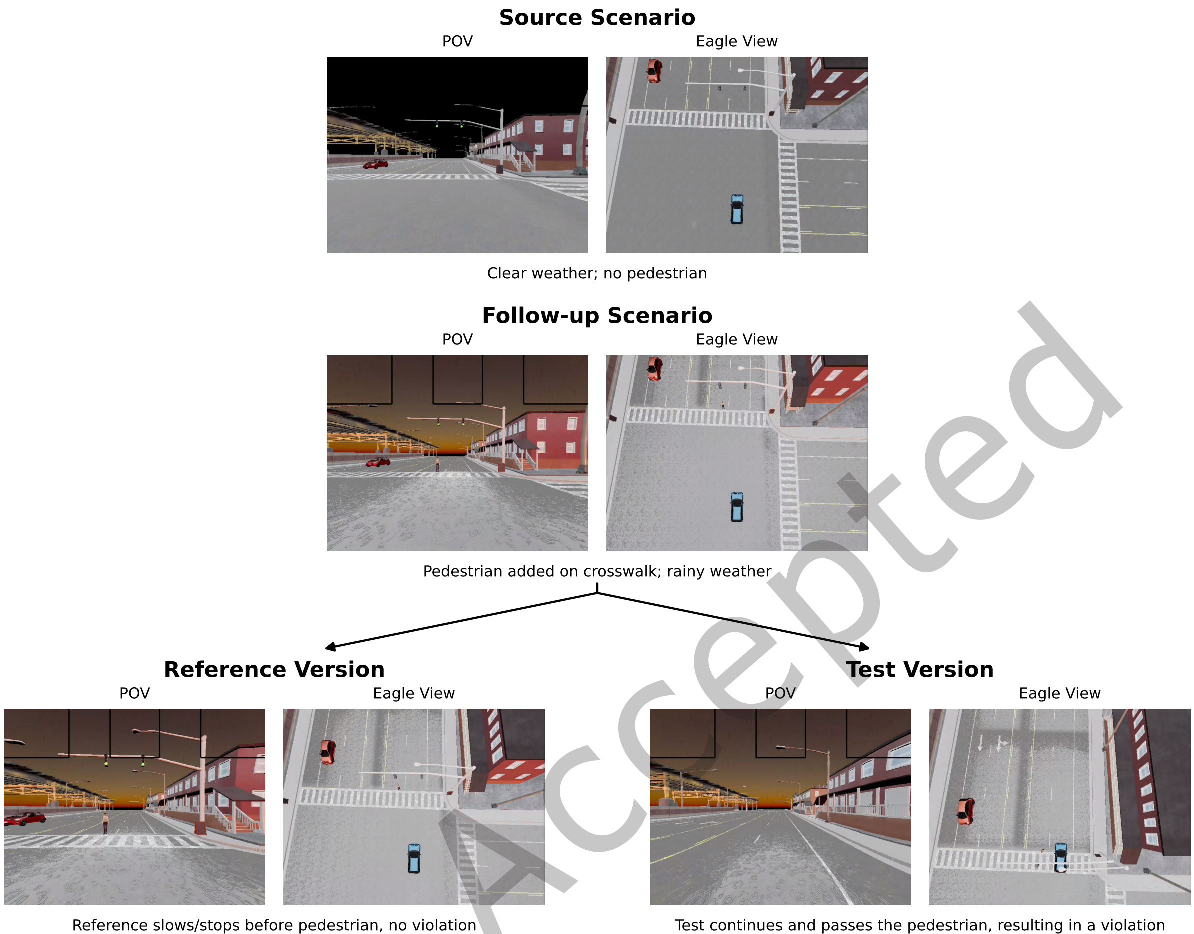

Fig. 5. Illustrative discrepancy revealed by *CoCoMagic*. The source scenario (top) depicts an ego vehicle traversing an intersection under clear weather conditions with no interfering agents. A follow-up scenario (middle) is generated by applying a metamorphic transformation that introduces a pedestrian in the crosswalk ahead of the ego vehicle and changes the weather to rainy. The follow-up scenario is then executed on two versions of the ADS: the reference (bottom-left) and the test (bottom-right). In the reference version, the vehicle slows down and stops at a safe distance from the pedestrian, satisfying the MR. In contrast, the test version continues without sufficient deceleration and passes the pedestrian at close proximity, resulting in a violation.

induce more severe behavioral discrepancies between the two ADS versions. Its ability to generate both critical and diverse test cases highlights the strength of its co-evolutionary search strategy in navigating complex search spaces and uncovering meaningful discrepancies, making it an effective framework for rigorous differential testing of ASs.

Answer to RQ1

CoCoMagic outperforms both *SGA* and *RS* in detecting severe differences in MR violations, as measured by the average *DS* metric. In addition, it achieves an average fitness that is 53% higher than *SGA* and 40% higher than *RS*, underscoring its effectiveness in uncovering more pronounced behavioral discrepancies.

7.2 RQ2: Efficiency of CoCoMagic

Fig. 6 presents the average *DS* values over simulation budget for each method, under nine different configurations of fitness thresholds θ_f and distance thresholds θ_d. Each subplot corresponds to a specific threshold setting, capturing lenient ($\theta_f = 1.0$), moderate ($\theta_f = 1.5$), and high ($\theta_f = 2.5$) severity requirements, each paired with increasing levels of diversity ($\theta_d = 1.0, 1.6, 2.2$). The distance thresholds were selected based on their filtering effect on the generated solutions. On average, $\theta_d = 1.0, 1.6$, and 2.2 remove about 10%, 30%, and 50% of the solutions produced by all methods. These values, therefore, correspond to low, moderate, and high diversity constraints.

Across nearly all configurations, *CoCoMagic* consistently outperforms the baseline methods in the total number of distinct solutions discovered as the simulation budget increases. Moreover, *CoCoMagic* exhibits a steeper growth trajectory compared to the baselines, which reflects the advantage of the co-evolutionary setting in promoting the discovery of additional distinct solutions as the budget increases. The advantage of *CoCoMagic* is particularly noticeable during the early stages of the search. In most configurations, the method begins discovering distinct solutions earlier than the baselines and accumulates them faster. This indicates higher search efficiency, as a larger number of useful discrepancies can be identified using a smaller portion of the available simulation budget.

As the fitness threshold becomes stricter ($\theta_f \in \{1.5, 2.5\}$), the number of retained solutions decreases for all methods, leading to lower overall *DS* values. This behavior is expected because fewer test cases satisfy the more demanding severity requirements. Nevertheless, *CoCoMagic* maintains a clear advantage over the others. For example, under moderate thresholds $\theta_f = 1.5$ and $\theta_d = 1.6$, *CoCoMagic* nearly doubles the number of discovered distinct solutions relative to both *SGA* and *RS* by the 100% budget mark.

When the diversity requirement becomes more restrictive ($\theta_d = 2.2$), the gap between the methods narrows in some configurations. In particular, under mild fitness constraints ($\theta_f = 1.0$), *SGA* becomes competitive in later stages of the search (budget $\geq$ 80%). However, this advantage of *SGA* is limited and largely disappears as the fitness requirement becomes stricter ($\theta_f = 1.5$ and $\theta_f = 2.5$). This indicates that while *SGA* may identify slightly more diverse but mild discrepancies, *CoCoMagic* is more effective at discovering severe and diverse discrepancies simultaneously, which is the primary goal of the testing process.

To quantitatively assess the overall efficiency of each method, we compute the *AUC* for the *DS*-budget curves shown in Fig. 6. This metric aggregates the progression of distinct solution discovery over the entire simulation budget, offering a single representative value for each configuration. A higher *AUC* indicates that a method finds more distinct solutions earlier and more consistently throughout the search process. On average, *CoCoMagic* outperforms *SGA* by 126% and *RS* by 169% in terms of *AUC*, demonstrating more efficient test case generation under budget constraints. Statistical analysis using the Wilcoxon signed-rank test [81] confirms the significance of these improvements, with p-value < 10^{-2} against both baselines.

In addition to evaluating the discovery rate of distinct solutions over budget, we also examined the computational efficiency of the algorithms. Fig. 7 presents the distribution of execution times (in hours) across 10 runs for *CoCoMagic*, *SGA*, and *RS*. The results show that *CoCoMagic* consistently requires less computational time, with an average of 7.44 hours, compared to 8.99 hours for *SGA* and 8.91 hours for *RS*. This indicates that *CoCoMagic* is not only more effective at utilizing the search budget but also faster to execute. Specifically, *CoCoMagic* improves average execution time by 17.24% over *SGA* (p-value < 10^{-2}) and by 16.49% over *RS* (p-value < 10^{-3}). The longer execution time observed for *SGA* and *RS* can be attributed to their tendency to generate more invalid complete

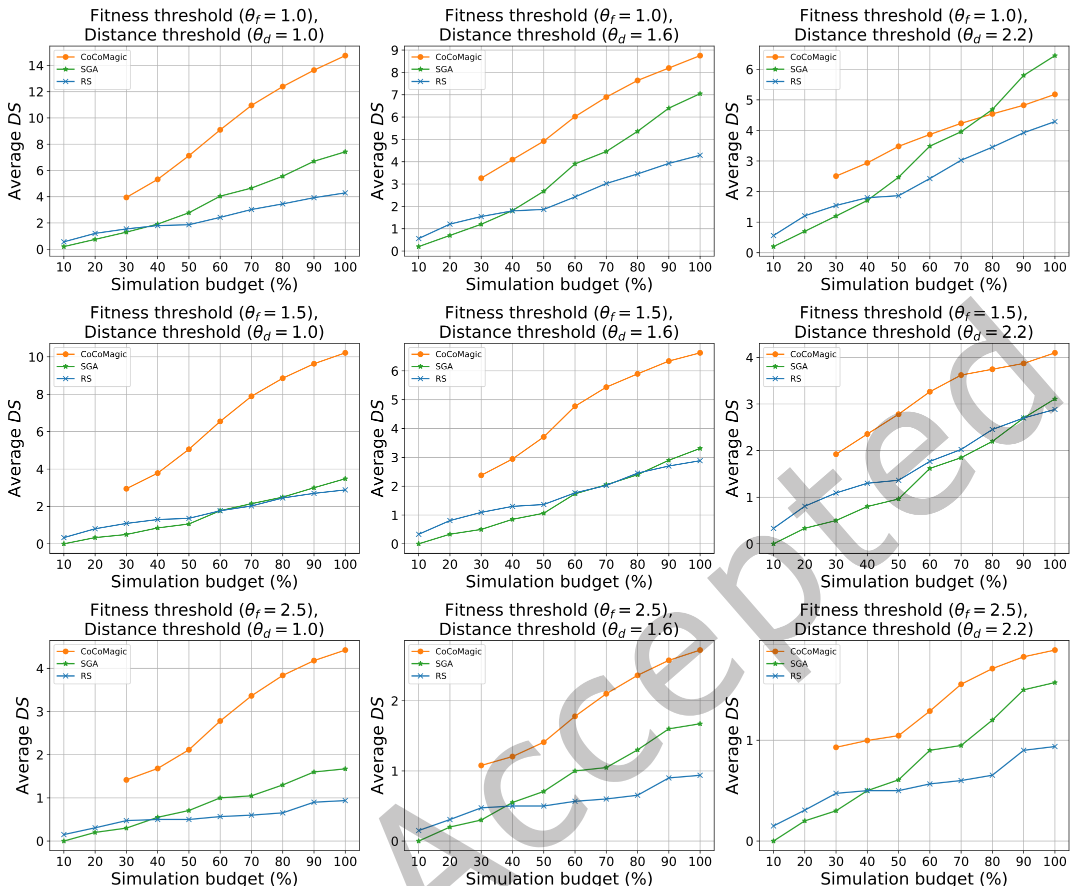

Fig. 6. *Distinct Solutions (DS)* progression across simulation budget levels, normalized to a total of 500 simulations. Each subplot illustrates how the *DS* value grows as more simulation resources are consumed, under specific combinations of fitness thresholds (θ_f) and distance thresholds (θ_d). The curves represent different methods, allowing a direct comparison of how quickly and effectively each approach uncovers severe and diverse behavioral discrepancies between ADS versions. Sharper increases and higher endpoints indicate greater efficiency in identifying test cases early in the search process.

solutions during the search. These invalid solutions violate simulation constraints, for example, by placing objects in overlapping locations, which causes the simulator to terminate immediately at the start of execution. Although such solutions are excluded from the simulation budget and discarded from the evolutionary process, the effort spent attempting to simulate them still incurs additional computational overhead, resulting in longer overall execution times.

Overall, these results demonstrate that *CoCoMagic* is significantly more budget-efficient at discovering critical and diverse test cases, while also achieving faster execution than the baselines. These advantages make *CoCoMagic* better suited for cost-sensitive testing scenarios.

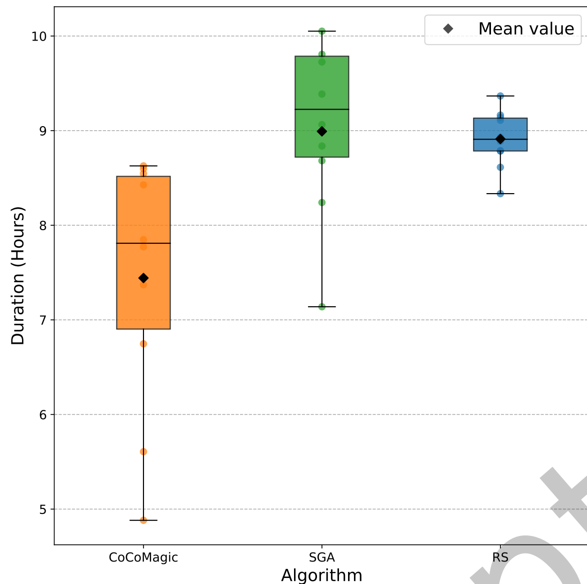

Fig. 7. Distribution of execution times (in hours) for *CoCoMagic*, *SGA*, and *RS* under a fixed search budget. Lower execution time indicates greater computational efficiency in completing the search. Each box summarizes the distribution of execution times across 10 runs.

Answer to RQ2

CoCoMagic outperforms *SGA* by 126% and *RS* by 169% in terms of overall efficiency, as measured by the *AUC* of the *DS*-budget curves, demonstrating its superior ability to discover critical and diverse test cases within a limited search budget. In addition, it achieves 17% faster execution time than *SGA* and 16.5% faster than *RS*, confirming its advantage in computational efficiency.

7.3 RQ3: Impact of Constraints and Population Initialization

Fig. 8 shows the *AED* between the generated complete solutions and the set of previously observed execution scenarios. This metric reflects how closely the identified solutions resemble behaviors observed during actual system executions. Lower *AED* values indicate that the generated complete solutions are behaviorally closer to known execution scenarios, whereas higher values suggest the discovery of more dissimilar solutions.

The most noticeable trend in Fig. 8 is the substantial reduction in *AED* values achieved by both *RI* configurations. *CoCoMagic with RI* produces the lowest *AED* values overall, followed by its unconstrained counterpart. This outcome confirms the effectiveness of *RI* in guiding the search toward solutions that remain close to observed ones. By seeding the population with scenarios observed during actual executions, *RI* biases the search toward regions of the search space that are more contextually grounded. In contrast, the unconstrained and constrained configurations without *RI* (i.e., *CoCoMagic\c* and *CoCoMagic*) yield significantly higher *AED* values, with distributions clustered around a much higher median. This confirms that, in the absence of *RI*, the search is more likely to generate complete solutions that deviate from what is commonly observed, resulting in less realistic but potentially more severe solutions.

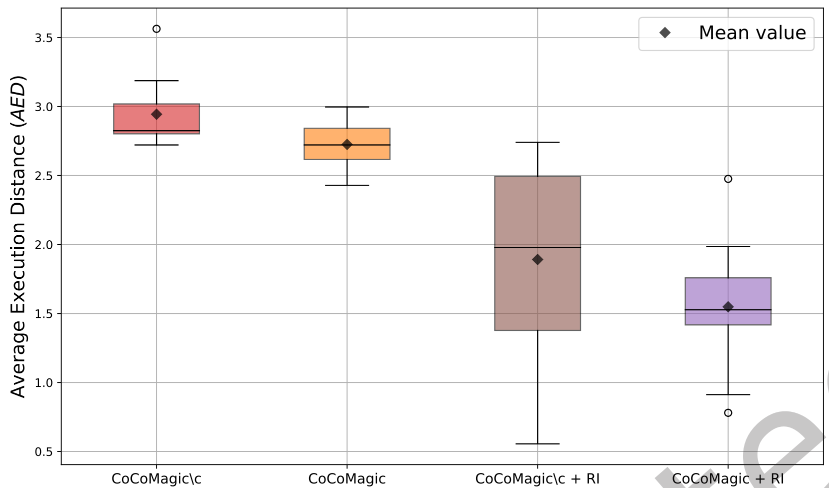

Fig. 8. *Average Execution Distance (AED)* between identified test cases and execution scenarios under different configurations of *CoCoMagic* and baseline methods. Higher *AED* values indicate that the generated test cases are more distinct from the execution scenarios. Each box summarizes the distribution of *AED* values for a given method, highlighting how closely the identified test cases resemble previously observed real-world scenarios.

The impact of the constraining mechanism is particularly evident when comparing constrained and unconstrained variants within each initialization strategy. In both cases, applying the constraint results in lower *AED* values, indicating that it effectively guides the search toward solutions closer to the observed execution scenarios. Quantitatively, applying the constraint alone without *RI* yields a modest improvement of 7.43% in average *AED* value (2.73) compared to its unconstrained counterpart (2.94). The benefit becomes more pronounced when *RI* is introduced. The unconstrained variant with *RI* reduces the average *AED* value by 35.76% (1.89) relative to the non-*RI* variant. When combining both *RI* and constraining, the search is tightly focused on execution scenarios, resulting in the most significant improvement: a 47.41% (1.55) reduction in *AED*. This highlights the complementary role of constraints in enhancing behavioral realism, especially when the initial population is already grounded in the observed execution scenarios. Overall, these results illustrate the effectiveness of both *RI* and the constraining mechanism in promoting the realism of generated test cases.

Fig. 9 compares the average number of distinct solutions discovered by each configuration of *CoCoMagic*, across varying distance thresholds (θ_d) and fitness thresholds (θ_f). Across the six fitness-threshold settings, *CoCoMagic* and *CoCoMagic\c* exhibit similar overall behavior, with only modest differences observed across combinations of θ_f and θ_d. For lower fitness thresholds ($\theta_f = 1.0, 1.3, 1.5$), *CoCoMagic* tends to achieve slightly higher average *DS* values than *CoCoMagic\c* at smaller distance thresholds. However, this gap is relatively small and diminishes as θ_d increases, with both configurations converging to comparable performance at larger distance thresholds. For stricter fitness thresholds ($\theta_f = 1.8, 2.0, 2.3$), a mild shift can be observed, where *CoCoMagic\c* slightly exceeds *CoCoMagic*. This suggests that, under stricter severity requirements, removing the constraint can help retain more solutions.

Introducing *RI* leads to a noticeable reduction in *DS* values, for both constrained and unconstrained variants. This reduction can be attributed to the tighter focus imposed by sampling initial populations from execution scenarios, which biases the search toward more realistic, but potentially more restricted regions of the search space. However, *RI* exhibits reduced sensitivity to the distance threshold θ_d; both *RI* variants maintain a more

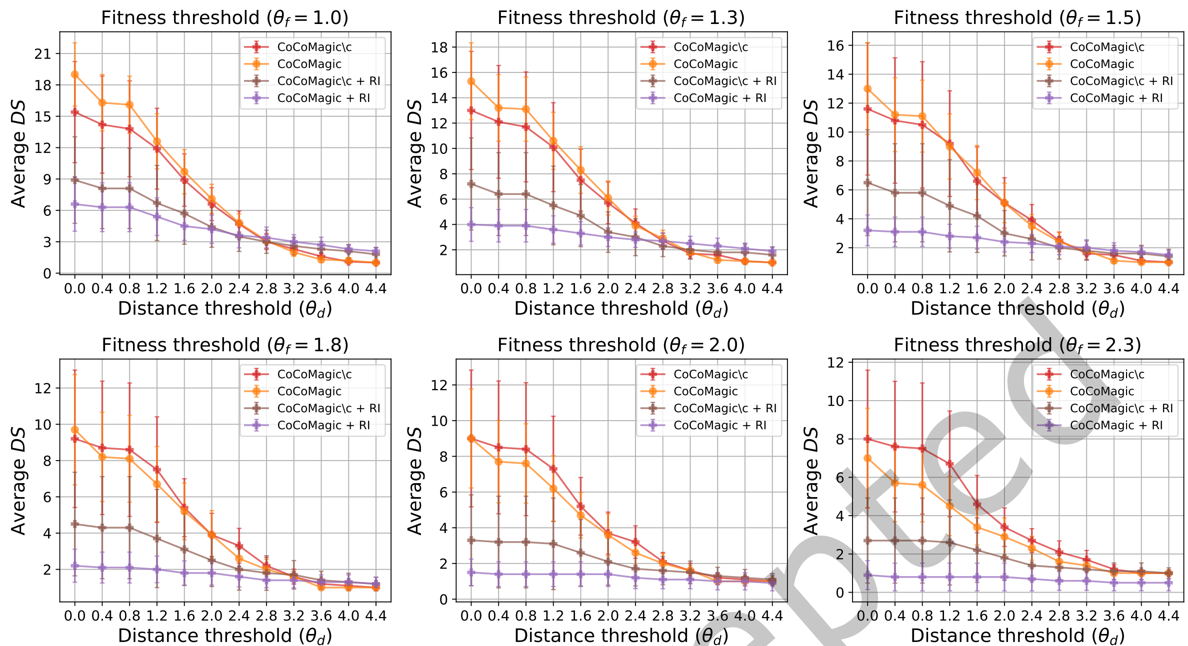

Fig. 9. *Distinct Solutions (DS)* vs. distance threshold (θ_d) across different fitness thresholds (θ_f) for different configurations of *CoCoMagic*. A higher *DS* value indicates more distinct test cases discovered by the method. Each curve plots the average *DS* value at different θ_d settings, under a specific fitness threshold θ_f. This figure reveals how each method balances the quantity and diversity of test cases as θ_f varies.

stable *DS* value compared to their counterparts, as θ_d increases. This robustness is likely due to the higher diversity inherent in the sampled execution scenarios used for initialization, which enables the generation of complete solutions that are already dispersed across different behavioral regions.

These trends are also reflected quantitatively in the overall reduction of *DS* values compared to the unconstrained, non-*RI* baseline (*CoCoMagic\c*). Applying the constraint alone results in a 3.97% (p-value = 0.028) decrease in the *DS* metric, highlighting the limiting effect of realism enforcement on exploratory diversity. Introducing *RI* to the unconstrained variant leads to a slightly larger reduction of 21.22% (p-value < 10^{-34}), while combining both constraints and *RI* produces the most substantial drop, with a 33.83% (p-value < 10^{-78}) decrease in *DS*. These results confirm the previously discussed trade-off: while constraints and *RI* improve behavioral realism, they reduce the algorithm's ability to discover a wide range of distinct solutions.

Fig. 10 presents the fitness distribution of solutions identified by each configuration, excluding those with insignificant fitness values (< 0.1), across 10 executions. The metric reflects the severity of the discovered solutions, with higher values indicating more critical discrepancies between the two ADS versions.

As shown in Fig. 10, the unconstrained variant (*CoCoMagic\c*) achieves the highest average fitness across runs, followed closely by the constrained variant. This observation may suggest that removing the constraint allows the search to better exploit regions of the search space that lead to more severe discrepancies. However, the small reduction in average fitness observed for *CoCoMagic* and *CoCoMagic\c+RI* relative to *CoCoMagic\c* is not statistically significant, suggesting that there is no solid evidence that introducing constraints or initialization with execution scenarios diminishes the ability to find severe discrepancies. In contrast, the reduction observed

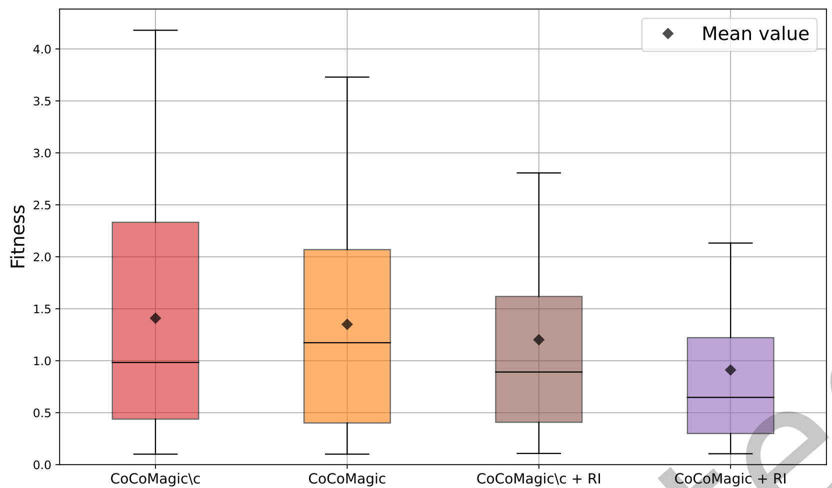

Fig. 10. Average fitness of test cases generated by different configurations of *CoCoMagic*. Higher average fitness indicates that the discovered test cases induce greater differences in the violations exhibited by the two ADSs. Each box shows the average fitness values of the generated test cases across 10 executions. This figure highlights the relative strength of each method in uncovering test cases with severe behavioral differences.

for *CoCoMagic+RI* compared to *CoCoMagic\c* is statistically significant. This indicates that when both constraints and initialization are applied together, the search is more strongly biased toward typical or nominal driving patterns, restricting access to rare edge-case behaviors that produce higher-fitness solutions.

Quantitative comparisons provide insights into the effect of constraints and *RI* on the severity of discovered solutions. Applying the constraint alone leads to a reduction of 4.20% in average fitness values, with the constrained configuration (*CoCoMagic*) achieving an average fitness of 1.35 compared to 1.41 for the unconstrained baseline (*CoCoMagic\c*). Similarly, incorporating *RI* into the unconstrained variant yields a 14.72% decrease, with an average fitness of 1.20. However, neither of these reductions is statistically significant ($p \approx 0.42$ and $p \approx 0.07$, respectively). In contrast, combining both constraints and *RI* produces a substantial and statistically significant reduction, with an average fitness of 0.91, corresponding to a 35.37% drop from the baseline ($p < 10^{-5}$).

Overall, these findings indicate that applying either constraints or *RI* alone can slightly reduce the severity of the generated solutions by limiting exploration. Yet, this effect is negligible and statistically insignificant. In contrast, combining the two results in a significant reduction in severity, demonstrating that their joint influence can substantially limit the search from reaching highly critical discrepancies.

In summary, the results of RQ3 reveal a clear trade-off between behavioral realism and exploratory power when applying constraints and *RI*. While both mechanisms independently and jointly reduce the number and severity of generated test cases, they significantly enhance alignment with real-world scenarios, as evidenced by lower *AED* values. *RI* proves especially effective in grounding the search, but it also significantly restricts access to edge-case solutions, particularly under stricter fitness requirements. Conversely, unconstrained and randomly initialized search enables broader exploration and uncovers more critical solutions, though at the cost of lower realism. These findings emphasize the importance of selecting the appropriate configuration based on the testing objective and of prioritizing realism, severity, or diversity in the discovered solutions.

Answer to RQ3

Both constraints and *RI* improve the behavioral realism of generated test cases, as measured by *AED*, with their combination yielding around 50% improvement over the unconstrained baseline. However, these mechanisms also tend to reduce the number of discovered solutions. The constraint alone leads to a 4% reduction in *DS* values. Incorporating *RI* further decreases *DS* values by up to 34%. This reveals a clear trade-off: constraints and *RI* enhance realism but may reduce the quantity and severity of the identified test cases.

7.4 RQ4: Interpretability Approach

7.4.1 Quantitative Evaluation of Interpretability. Fig. 11 presents the distribution of *MAE* values for predictions made by *RuleFit* in *CoCoMagic* and the baseline methods, *RF* and *GBDT*. The results indicate that *RuleFit* achieves an average *MAE* value of 0.71, which is higher than that of *RF* (0.68) and *GBDT* (0.67). Although *RF* and *GBDT* yield slightly lower average *MAE* value, indicating higher accuracy in modeling behavioral differences in MR violations, they lack interpretability as they tend to generate a large number of complex rules. In decision tree-based models like *RF* and *GBDT*, these rules correspond to decision paths, which are sequences of feature-based conditions followed from the root to a leaf node in each tree. On average, *RF* produces 1009.95 such paths and *GBDT* produces 2010.00, making it difficult to extract clear, human-understandable insights from their predictions. This limitation reduces their practical usefulness for understanding the behavioral discrepancies between system versions. In contrast, *RuleFit* strikes a balance between accuracy and interpretability, producing a concise set of human-readable rules (28.72 rules on average) that facilitate understanding while still achieving accuracy close to that of *GBDT* and *RF*. Furthermore, the rules generated by *RF* and *GBDT* are more complex, averaging 3.91 and 4.10 conditions per rule, respectively, whereas *RuleFit* rules average only 2.09 conditions. This simplicity enhances the interpretability of *RuleFit* rules, making them more accessible for human analysts.

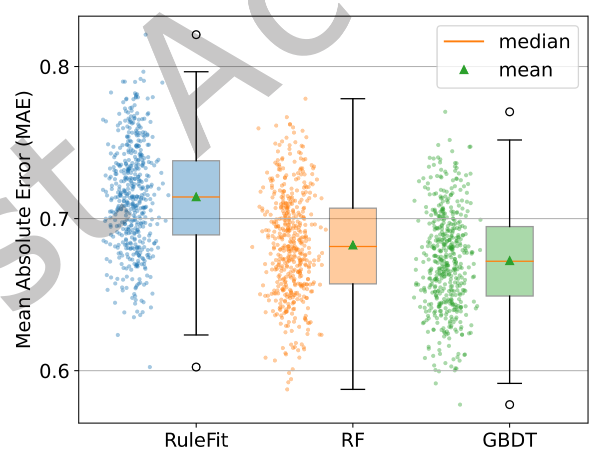

Fig. 11. Distribution of *Mean Absolute Error (MAE)* for *RuleFit* in *CoCoMagic* and baseline methods. A lower *MAE* value indicates greater model accuracy in capturing behavioral differences between the two system versions. Individual points beside each box represent the *MAE* values for each identified test case.

The *support* distribution demonstrates that the rules extracted by *RuleFit* consistently cover all of the identified complete solutions. The generated rules achieved 100% *support*, meaning every identified complete solution was covered by at least one rule. Given the maximum of 30 rules allowed per run, achieving complete coverage of all identified solutions is particularly noteworthy. This high level of *support* underscores the effectiveness and consistency of *RuleFit* in capturing a compact yet comprehensive set of interpretable rules or patterns that explain the observed behavioral differences across diverse runs.

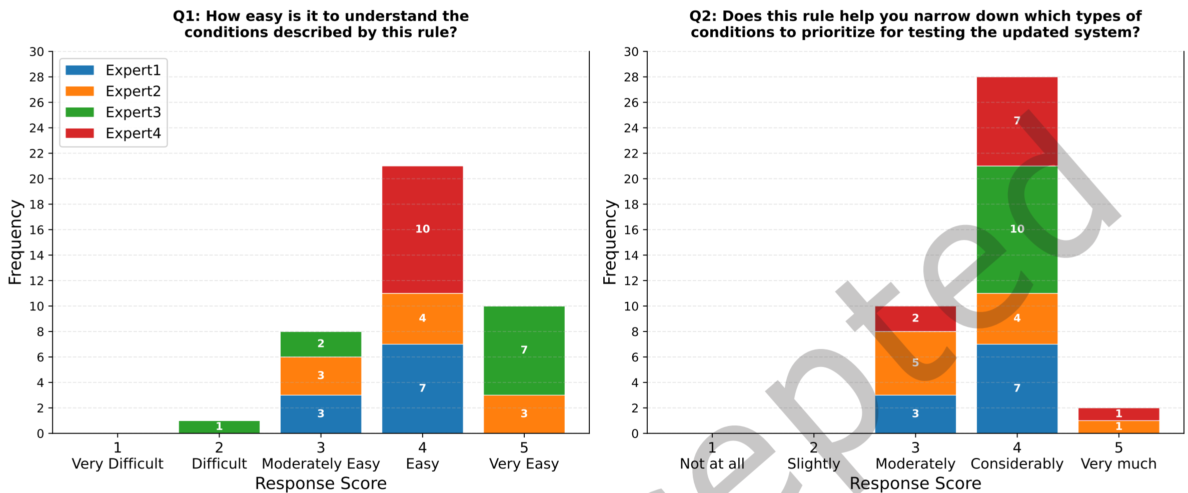

Fig. 12. Distribution of expert ratings for the generated rules across two evaluation criteria: ease of understanding (*Q*1), and usefulness for prioritizing testing conditions (*Q*2). Ratings are based on a five-point Likert scale.

7.4.2 Expert Assessment of Interpretability Rules. Fig. 12 shows the distribution of expert ratings for the generated rules across the two evaluation questions. The results indicate that the rules are generally understandable and practically useful.

The responses to *Q*1 show a clearly positive pattern. Across the 40 ratings collected for this question (10 rules × 4 experts), the average score is 4.0, which corresponds to the *Easy* response. In total, 31 out of 40 ratings (77.5%) are either *Easy* or *Very Easy*, while only a single rating falls below *Moderately Easy*. This indicates that the conditions described by the rules are largely understandable to practitioners, and experts rarely found the rules confusing or difficult to interpret. The results are also relatively consistent across experts. In most cases, the ratings assigned to a given rule differ by at most 1 point. Larger disagreements are rare, with only two rule assessments showing a difference of 2 points between experts.

A similar trend can be observed for *Q*2. Out of the 40 ratings, 30 (75.0%) are *Considerably* or *Very much*, and none of the responses fall in the two lowest categories. The mean score is 3.8, lying between *Moderately* and *Considerably* useful and closer to the latter. This suggests that, on average, the experts found the rules useful for narrowing down which conditions to prioritize when testing the updated system.

Overall, the questionnaire results indicate that the generated rules are understandable to domain experts and useful for narrowing the testing focus to relevant driving conditions. These findings support the claim that the interpretability component can help engineers move from raw test cases toward more actionable testing insights.

To further illustrate the practical value of the interpretability approach, Table 2 presents several representative rules from one run, together with their associated performance changes. Each rule characterizes a specific driving condition under which the two ADS versions exhibit different MR violations. The *Driving Condition Description*

Table 2. Automatically generated rules describing scenarios that induce behavioral differences between ADS versions.

Driving Condition Description	Performance Change
The ego vehicle is moving at a slow speed, and a pedestrian on the left is extremely close.	Degradation
Pedestrians ahead of the ego vehicle are very close and all are moving.	Degradation
Pedestrians ahead move along the road direction while a left-side pedestrian is close.	Degradation
A left-side pedestrian is moving toward the right.	Improvement
Ego vehicle is moving slowly under very dark lighting conditions.	Improvement
There are slow-moving vehicles on the left, and the front vehicles are oriented to the right.	Degradation
The ego vehicle is moving while a static obstacle ahead is extremely close.	Degradation

column reports natural-language descriptions derived from *RuleFit* rules via the post-processing step described in Section 6.7.4, while the *Performance Change* column indicates whether the observed divergence corresponds to a degradation or an improvement with respect to GP_1. This is inferred from the sign of the coefficient associated with the rule in *RuleFit*, where a positive coefficient indicates a degradation, and a negative coefficient indicates an improvement. For instance, the first rule indicates that when the ego vehicle is moving at a low speed while a pedestrian on the left is in very close proximity, the updated version tends to exhibit more severe MR violations. These rules are not meant to be interpreted as definitive diagnostic conclusions. Instead, they provide actionable starting points for identifying driving conditions that warrant further investigation when analyzing behavioral differences between system versions.

In summary, the results demonstrate that *RuleFit*, as used in *CoCoMagic*, effectively balances accuracy and interpretability in modeling behavioral differences in MR violations. It produces a compact set of human-readable rules with competitive accuracy, indicating a reliable ability to capture behavioral discrepancies. It also provides practical insights into whether and under which conditions the updated system exhibits more violations than the original, supporting informed decisions about real-world deployment. This approach provides interpretable, actionable explanations of the factors driving behavioral discrepancies, thereby facilitating a more comprehensive understanding of system updates.

Answer to RQ4

In our interpretability approach, *RuleFit* effectively balances accuracy and interpretability when modeling differences in MR violations, producing a compact set of human-readable rules with accuracy comparable to *GBDT* and *RF*. Additionally, the expert assessment confirms that the generated rules are generally understandable and practically useful, particularly for guiding testing efforts and identifying relevant conditions for further investigation.

8 Discussion

In this section, we discuss several important aspects of our proposed framework, including the trade-offs involved in using constraints and population initialization strategies, the applicability of our approach to other types of ASs, practical considerations, and potential threats to the validity of our results.

8.1 Trade-offs in Using Constraints and the Population Initialization Strategy

In search-based generation of realistic and effective test cases, the use of constraints and the choice of population initialization strategy are critical factors shaping the realism, severity, and quantity of the resulting scenarios. These two factors influence the balance between producing test cases that closely reflect real-world conditions and exploring a wide range of potentially risky or severe behavioral discrepancies. Achieving this balance inevitably involves trade-offs. Tighter controls or heavy reliance on observed execution scenarios may improve realism but can limit the discovery of critical edge cases, whereas looser controls may enhance quantity but may reduce practical relevance.

Constraints guide the search toward regions of the solution space that are more relevant to real-world conditions. By penalizing unrealistic or infeasible conditions, they ensure that the generated test cases remain grounded in scenarios that could occur in practice. This targeted focus enhances the practical value of identified test cases by reducing time spent analyzing irrelevant or impossible cases. However, collecting a diverse and representative set of execution scenarios is important to capture the breadth of conditions under which failures may occur. Excessive similarity among collected execution scenarios can cause the testing process to miss distinct failure modes that may arise under different yet equally plausible conditions. Moreover, overly restrictive constraints can narrow the search space excessively, omitting rare but potentially critical situations that might expose severe behavioral discrepancies. Conversely, relaxing the constraints expands exploration and increases the diversity of generated test cases, but at the cost of admitting unrealistic or invalid scenarios that require additional computational filtering. Based on our empirical findings, a constraint level that penalizes approximately 60–70% of candidate test cases per generation tends to strike an effective balance, maintaining both solution realism and sufficient severity.

Similarly, the population initialization strategy introduces its own set of trade-offs. Initializing the search with execution scenarios sampled from real-world data (i.e., the *RI* strategy) accelerates convergence by starting from scenarios already representative of operational contexts. This often yields test cases that are immediately applicable and realistic, making the approach well-suited when computational budgets or testing time are constrained. However, the drawback is that reliance on known scenarios can bias the search toward familiar behaviors, limiting the discovery of novel or unexpected situations that have not been previously observed but may reveal critical behavioral discrepancies. In contrast, a purely random initialization promotes broader exploration, improving the likelihood of uncovering rare, high-severity cases that challenge system safety or reliability. The trade-off is that such cases may be less realistic and, without a sufficiently diverse and complete set of representative execution scenarios, the search may overlook critical, realistic conditions that could pose safety risks.

Selecting the right balance between constraints and initialization strategies depends on the specific objectives of the testing. Our experimental results (see Section 7.3) show that combining constraints with *RI* produces test cases with the highest realism. However, this approach tends to limit the discovery of the most severe behavioral discrepancies. Therefore, practitioners must weigh the trade-offs carefully, tailoring their approach to ensure both targeted realism and adequate exploration to effectively identify potential behavioral discrepancies and safety issues.

8.2 Applicability to Other Autonomous Systems

Although our evaluation focuses on autonomous driving, the proposed differential, search-based testing workflow is generic and transferable to a wide range of ASs. The core components of the framework, including the search algorithms, differential analysis logic, and constraining mechanisms, operate independently of any specific AS domain. All domain assumptions are encapsulated in a small set of plug-in modules, allowing practitioners to adapt the framework by implementing only domain-specific interfaces without modifying the generic core.

In general, adapting the framework to a new domain requires the practitioners to define three domain-specific components that interface the generic core with the semantics of the target AS. As a hypothetical illustrative example, consider a robotic assembly system in which a controller coordinates a robotic arm to assemble components in an assembly line. In this setting, first, the practitioners specify a set of domain-relevant MRs that encode expected behavioral consistency under controlled changes to the environment. For instance, an MR *or* may state that increasing the assembly-line speed by x should not increase the component failure rate by more than y.

Based on the selected MRs, practitioners define a scenario representation that captures the characteristic parameters of a scenario, including those subject to change by the selected MRs. A simple representation can be a vector whose dimensions capture key operating conditions (e.g., the speed of the assembly line and the position of items on the line). Second, the practitioners implement the scenario execution module, which runs a given scenario, typically using a simulator, and returns the metrics required by the MRs (e.g., the failure rate). Finally, for each specified MR, the practitioners define how a follow-up scenario is derived from a source scenario through perturbations (e.g., increasing the speed parameter by x), and specify the corresponding output relation that maps the observed outcomes to a continuous violation severity score, i.e., extent of MR violation (see Eq. (1)). In this example, the function D_{or} that quantifies the deviation from *or* could be defined based on the difference in failure rates between the source and follow-up scenarios. For instance, such a function can be defined as $D_{or} := r_{\text{follow-up}} - (1 + y) \times r_{\text{source}}$, where r_{source} and $r_{\text{follow-up}}$ denote the observed failure rates in the source and follow-up scenarios, respectively, and the result of $D_{or} > 0$ indicates an MR violation. Once these domain-specific modules are in place, they can be integrated into the framework, allowing the core workflow to operate directly on the specified domain without further modification.

8.3 Practical Considerations

In this section, we discuss three practical considerations regarding the scalability of our approach, the significance of distinct solutions in the context of AS safety, and the interpretation of differences in MR violations.

8.3.1 Scalability and Computational Cost. In terms of scalability, the dominant cost in our workflow is scenario execution rather than the generic search and differential analysis logic. The overhead of population management, constraint handling, differential evaluation, and diversity management is comparatively small as these steps operate on compact scenario representations and scalar violation scores. As a result, the scalability of the overall approach is primarily driven by the throughput of the underlying simulator or testbed, the available computational resources, and the extent to which scenario executions can be distributed across parallel simulation instances within a given simulation budget. This reflects a common characteristic of search-based testing for ASs, where each additional candidate solution requires an expensive end-to-end execution to obtain safety-relevant outcomes.

Given this cost structure, search efficiency translates directly into scalability and cost-effectiveness. Since our framework uncovers severe differential violations with fewer executed scenarios than the baselines, it reduces the number of expensive simulator runs required to reach a fixed testing objective. In addition, the framework explicitly supports parallel execution of scenarios by dispatching independent evaluations to multiple simulation agents. Since each candidate scenario can be executed and scored independently, increasing the number of concurrent simulator instances directly improves throughput and can substantially reduce wall-clock testing time within the same simulation budget. Together, these properties make the framework well-suited to iterative development settings, where timely feedback on update-induced behavioral degradations is essential, and where testing throughput is constrained primarily by simulation capacity rather than by the optimization logic itself.

8.3.2 Practical Significance of Distinct Solutions. As we use *DS* as a primary measure of effectiveness and efficiency, it is important to clarify its practical significance in the context of AS safety. Each distinct solution corresponds to a pair of source and follow-up scenarios that yield different outcomes in terms of MR violations, indicating behavioral divergence between system versions. In practice, such divergences may correspond to safety-critical situations, depending on the application domain. For example, in autonomous driving, these differences may indicate collisions, near-misses, or traffic rule violations. In other ASs, they may lead to mission failures, unsafe interactions, or violations of operational constraints.

Distinct solutions indicate that the testing approach is exploring different regions of the operational space, including varied environmental conditions and interactions with external agents. A higher number of distinct solutions, particularly when they are sufficiently dissimilar, suggests that the testing process is uncovering a broader and more diverse set of critical scenarios and edge cases that could compromise system safety. An AS that exhibits fewer unexpected behavioral differences across such a wide range of scenarios provides stronger evidence of robustness and reliability. Conversely, the discovery of many distinct behavioral differences highlights areas where the system may require additional analysis, targeted testing, or refinement.

Overall, the number of distinct solutions exposing behavioral differences serves as a practical proxy for how thoroughly an AS has been challenged under safety-relevant conditions. The more distinct solutions identified, the greater the insight gained into potential safety risks and system vulnerabilities.

8.3.3 Interpretation of Differences in MR Violations. The behavioral differences identified by *CoCoMagic* are expressed through differences in MR violations across system versions. These differences are not intended as a measure of absolute system correctness, but rather as a confirmation of improvement or degradation with respect to the particular safety property specified by the MR. When the original version satisfies an MR and the updated version violates it, the updated version demonstrably fails to preserve the expected behavioral relationship, thereby constituting a degradation of that property. This does not imply that the original output is correct in an absolute sense, but rather that the updated version violates a necessary safety property that the original version satisfied. Conversely, a transition from violation to satisfaction indicates improvement in that property, while other undetected faults may persist. Importantly, although MR satisfaction does not guarantee the absence of all faults, an MR violation does confirm the presence of a safety-relevant problem, namely that the system fails to exhibit the expected behavioral relationship specified by the MR.

Based on this interpretation, differences in MR violations signal potential degradations warranting further investigation, or improvements that can provide confidence in the update. The test cases and interpretable rules generated by *CoCoMagic* provide practitioners with concrete, actionable evidence of how the updated system differs from its predecessor and under which conditions such divergences occur, supporting informed assessment. Whether such differences warrant blocking a release ultimately requires human judgment, as it depends on domain-specific factors such as the severity of the observed failures and the practitioners' risk tolerance. Accordingly, the outputs of *CoCoMagic* are designed to serve as complementary evidence that practitioners can incorporate into their broader decision-making process, alongside other quality assurance activities.

8.3.4 Guidance on selecting MRs. The selection of MRs can influence the performance of CoCoMagic, both through the number of MRs included in a search process and through the types of MRs selected. Regarding the number of MRs, using multiple MRs in a single search process can improve search effectiveness when the MRs share the same output relation. In CoCoMagic, a search process is guided by a fitness function that quantifies the difference in the extent of MR violation between two system versions. Therefore, grouping several MRs is meaningful only when their violations can be measured using the same output relation. In such cases, the search can explore a richer set of input perturbations while still relying on a coherent fitness signal. In our case study, the MRs in $GP1$ all expect a speed reduction, while the MRs in $GP2$ all expect invariance of the steering angle under their corresponding perturbations. Grouping such MRs allows the perturbation population to include multiple types of environmental changes while evaluating them against the same expected steering behavior. This can help the search generate potentially more complex follow-up scenarios, thereby increasing the chance of identifying changes that induce behavioral differences between the two system versions. In contrast, grouping MRs with incompatible output relations in the same search process can make the fitness signal less meaningful, since different relations may require different violation functions, thresholds, or output variables.

The type of MR is also important. CoCoMagic is most suitable for MRs that satisfy three practical requirements. First, the MR should define a relation between two test inputs, namely a source scenario and a follow-up scenario. Some MRs may involve three or more related inputs. Such MRs cannot be directly represented as a pair consisting of one input perturbation and one output relation, and therefore they are not currently supported by our framework without modification. Second, the input perturbation associated with the MR should be well-defined and feasible within the selected simulator or testing environment. Although CoCoMagic is not tied to a specific simulator, our case study uses Carla; therefore, the selected MRs had to be implementable within Carla's capabilities. For example, an MR requiring the replacement of buildings with trees would be difficult to apply in our setting if the simulator does not support such map-level modifications. More generally, whether Carla or another simulator is used, the MR's input perturbation must be executable through the available scenario configuration and simulation APIs.

Third, the output relation should be sufficiently well-defined to support a numerical measure of the extent of violation. This requirement is central to CoCoMagic because the search process is guided by a fitness value. If the expected output relation is ambiguous or purely qualitative, defining a numerical violation signal may become arbitrary. For example, an MR stating that the ADS should "drive more cautiously" after a perturbation is difficult to use directly unless cautiousness is operationalized through measurable outputs such as speed reduction, acceleration bounds, braking intensity, minimum distance to obstacles, or lane deviation. In our framework, we consider three common types of output relations: decreasing, increasing, and invariance relations. A decreasing relation expects the value of an output variable in the follow-up scenario to be lower than its value in the source scenario by a specified threshold. An increasing relation expects the opposite, namely that the follow-up output should be higher than the source output by a specified threshold. An invariance relation requires that the output difference between the source and follow-up scenarios remain within an acceptable tolerance. For each relation type, the extent of violation can be computed by measuring how much the observed outputs deviate from the expected relation. This numerical formulation enables CoCoMagic to guide the search toward test cases that reveal stronger violation differences.

Based on these considerations, we recommend selecting MRs that are pairwise, executable, and measurable. When multiple MRs share the same output relation, grouping them can be beneficial because it expands the space of meaningful input perturbations while preserving a coherent fitness definition. When MRs differ substantially in their output relations, they should preferably be evaluated in separate search processes or require a carefully designed multi-objective formulation.

8.4 Impact of Re-evaluation on False Alarms

To assess how many reported divergences may be artifacts of simulation instability, we analyzed the re-evaluation histories recorded for the evaluated solutions. We define a *false positive* as a candidate divergence whose initial execution exceeds the re-evaluation threshold of 0.2, but whose value after repeated execution and median aggregation falls below that threshold. This operational definition captures false alarms caused by nondeterminism during simulation.

For each evaluated solution, we reconstructed the initial DT divergence as the absolute difference between the initial MR violation values observed for the reference and test versions. We then compared this value with the final divergence after re-executing threshold-exceeding cases three times and aggregating them using the median. Table 3 reports the results for the primary CoCoMagic configuration.

Overall, 359 of the 412 initially threshold-exceeding CoCoMagic alarms remained above the threshold after re-evaluation, while 53 were removed. Thus, the estimated false-positive rate is 12.9% for the primary CoCoMagic configuration. Across all experimental runs, including baselines and ablations, 1,581 candidate alarms initially exceeded the threshold, 1,362 remained above it after re-evaluation, and 219 were removed, yielding a similar rate

Table 3. Impact of re-evaluation on false positives for the primary CoCoMagic configuration. *Initial* is the number of candidate alarms initially above the re-evaluation threshold, *Retained* is the number retained after median aggregation, *Dropped* is the number dropped below the threshold or discarded as invalid, and *FPR* is the number of dropped candidates divided by the initial candidates.

MR set	Evaluated	*Initial*	*Retained*	*Dropped*	*FPR*
*GP*1	1,068	337	296	41	12.2%
*GP*2	1,555	75	63	12	16.0%
Total	2,623	412	359	53	12.9%

of 13.9%. At the version-specific MR level, the corresponding rates were 11.0% for CoCoMagic and 10.5% across all configurations and baselines. These results indicate that re-evaluation removes a significant number of unstable alarms while preserving the large majority of reproducible behavioral divergences. Of course, such percentages are expected to vary across simulators, but our approach to eliminating false positives is always applicable.

The dropped cases were mainly attributable to nondeterministic simulator outcomes, a situation we expect to be common. We therefore keep re-evaluation as part of the reporting pipeline to reduce runtime-induced false alarms before engineers inspect the generated test cases.

8.5 Threats to Validity

This section discusses potential threats to the validity of our results, following the classification into four categories: Internal, External, Construct, and Conclusion validity [84].

8.5.1 Internal Validity. Internal validity reflects how reliably a study establishes a cause-and-effect relationship between the experiment and its outcomes. One potential threat is the sensitivity of population-based methods (i.e., *CoCoMagic* and *SGA*) to hyperparameter settings. The chosen configurations may not be optimal, potentially affecting performance. To address this, we adopted widely recommended hyperparameter values from the literature [62]. For parameters lacking established defaults, we selected values based on a pilot experiment.

Another internal validity concern arises from slight mismatches between the actual number of simulations executed and the allocated simulation budgets for the population-based methods. We mitigated this issue by applying linear interpolation to align simulation budgets across different methods. However, interpolation assumes a linear, continuous relationship between the metrics and the search budget. This assumption may not hold if metrics exhibit abrupt or non-linear changes across budgets, which could lead to inaccuracies in the aligned results. To reduce this risk, we validated the interpolation by analyzing metric trends and confirming that the linear assumption was reasonable within the tested budget range.

A further threat involves the correctness and reliability of the MRs used to detect behavioral differences. Since MRs serve as the basis for identifying violations, any inaccuracies, such as overly strict conditions, could result in false positives or negatives. To address this, we used well-established MRs from prior studies and conducted a pilot experiment to validate them. During this phase, we analyzed unexpected results and adjusted parameters to ensure the MRs were calibrated correctly. This refinement process helped ensure the reliability of the MRs in the main experiments.

8.5.2 External Validity. External validity concerns the extent to which results can be generalized to other contexts, particularly their applicability beyond the specific simulation environment used in this study. Due to the enormous effort and computational time required to prepare and run our experiments, a primary threat to external validity arises from our reliance on a single ADS (InterFuser) and a specific simulator (Carla) for the case study. For example, the substantial engineering effort required to enable seamless scenario execution using Carla and InterFuser constrained our ability to include additional platforms. While this choice may limit generalizability, both technologies are widely recognized within the autonomous driving research community. Carla is an open-source, high-fidelity simulator extensively used in academia and industry, and InterFuser ranked among the top-performing systems on the Carla *Leaderboard* during the evaluation period.

Another potential threat lies in the representativeness of the execution scenarios used. If the scenarios do not sufficiently reflect the variability of real-world driving conditions, the findings may lack general applicability. To mitigate this concern, we leveraged a variety of maps contributed by the Carla community, encompassing diverse urban layouts, road geometries, and traffic conditions. We executed the ADS in these environments and systematically collected execution scenarios at five-second intervals to ensure coverage of different driving situations, including variations in traffic density, weather, and road types. From the collected execution scenarios, we randomly sampled 800 scenarios, distributed equally across all maps, for use in our experiments. This sampling strategy aimed to ensure broad scenario diversity and better reflect the range of conditions encountered in real-world autonomous driving, thereby improving the generalizability of our results.

Another threat to external validity concerns the limited set of MR groups considered in our empirical evaluation. In this study, we evaluated CoCoMagic using two MR groups, $GP1$ and $GP2$, that capture common, practically relevant output-relation patterns in ADS testing. However, these groups do not cover all possible MRs that may be proposed in the future. Therefore, our empirical results should not be interpreted as evidence that CoCoMagic will perform equally well for every possible future MR category. Nevertheless, CoCoMagic is not inherently restricted to $GP1$ and $GP2$. The method only requires that an MR be expressible through two components: (i) an input perturbation, which defines how a source scenario is transformed into a follow-up scenario, and (ii) an output relation, which defines how the outputs of the two system versions should be compared and how the extent of violation should be measured. Once these components are specified, CoCoMagic can use the corresponding perturbation operators to generate follow-up scenarios and compute the differential fitness between system versions. Thus, extending CoCoMagic to new MRs mainly requires implementing the associated scenario transformation and defining a measurable violation function for the expected output relation, while the core co-evolutionary search, differential fitness computation, constraint mechanism, and initialization strategy remain unchanged.

8.5.3 Construct Validity. Construct validity concerns the extent to which the measurements used in a study accurately reflect the theoretical constructs they are intended to represent. In our work, a central concern is whether the *DS* metric appropriately captures each method's effectiveness in identifying differences in MR violations and in exploring diverse regions of the search space. Although this metric is designed to reflect these aspects, it may not fully encompass all dimensions of solution quality or diversity.

8.5.4 Conclusion Validity. Conclusion validity refers to the extent to which a research conclusion can be trusted. A notable limitation in our experiments is the relatively small number of repetitions, i.e., 10 runs per method, due to the substantial computational costs involved. To mitigate the risk of statistical error due to this small sample size, we report descriptive statistics along with their corresponding confidence intervals. This approach helps convey both the central trends and the uncertainty in the results, providing a more reliable basis for interpreting our findings.

9 Conclusion

In this paper, we presented *CoCoMagic*, a novel automated testing method that integrates Differential Testing (DT), Metamorphic Testing (MT), and advanced search-based testing to assess the overall impact of updates on AS behavior. By leveraging MT, *CoCoMagic* effectively identifies test cases that expose behavioral discrepancies between system versions, which may lead to safety violations. Its cooperative co-evolutionary approach efficiently explores the high-dimensional search space, directing the search toward test cases most likely to reveal significant behavioral differences. The use of constraints and population-generation strategies favors test cases that reflect realistic operating conditions and remain applicable to practical scenarios. The interpretability approach we propose complements this process by providing clear insights into the underlying causes of identified discrepancies, enabling developers to better understand and mitigate potential safety risks.

Our evaluation on the CARLA simulator with the INTERFUSER ADS shows that *CoCoMagic* consistently outperforms baseline methods across diverse budget and threshold settings, efficiently generating severe test cases that uncover important behavioral differences between system versions. These findings indicate that *CoCoMagic* offers an effective and efficient solution for AS differential testing in open contexts by decomposing the high-dimensional search space through cooperative co-evolution. The realism of the generated scenarios, combined with the interpretability approach's explanatory power, makes *CoCoMagic* a practical and actionable tool for developers aiming to improve the safety and reliability of evolving ASs.

Data Availability

The replication package for our experiments, which includes the implementation of our approach, baseline methods, configuration files, questionnaire, and experimental results, is available on Figshare at https://figshare.com/s/5d35bd01c27169e5ef2e.

Acknowledgments

This work was supported by a research grant from Huawei Technologies Canada Co., Ltd, as well as the Canada Research Chair and Discovery Grant programs of the Natural Sciences and Engineering Research Council of Canada (NSERC). Lionel C. Briand's contribution was partially funded by the Research Ireland grant 13/RC/209.

References

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE '16)*. Association for Computing Machinery, New York, NY, USA, 63–74. doi:10.1145/2970276.2970311

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE '18)*. Association for Computing Machinery, New York, NY, USA, 143–154. doi:10.1145/3238147.3238192

[3] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2020. Automated repair of feature interaction failures in automated driving systems. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20)*. Association for Computing Machinery, New York, NY, USA, 88–100. doi:10.1145/3395363.3397386

[4] Baidu. 2025. *Apollo*. https://en.apollo.auto/apollo-self-driving

[5] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, null (Feb. 2012), 281–305.

[6] Christian Birchler, Nicolas Ganz, Sajad Khatiri, Alessio Gambi, and Sebastiano Panichella. 2023. Cost-effective simulation-based test selection in self-driving cars software. *Science of Computer Programming* 226 (March 2023), 102926. doi:10.1016/j.scico.2023.102926

[7] Christian Birchler, Sajad Khatiri, Pouria Derakhshanfar, Sebastiano Panichella, and Annibale Panichella. 2023. Single and Multi-objective Test Cases Prioritization for Self-driving Cars in Virtual Environments. *ACM Transactions on Software Engineering and Methodology* 32, 2 (April 2023), 1–30. doi:10.1145/3533818

[8] Larry Bull. 1997. Evolutionary Computation in Multi-Agent Environments: Partners. In *Proceedings of the 7th International Conference on Genetic Algorithms* (East Lansing, MI, USA). Morgan Kaufmann, 370–377.

[9] Larry Bull. 1998. *Evolutionary computing in multi-agent environments: Operators.* Springer Berlin Heidelberg, Berlin, Heidelberg, 43–52. doi:10.1007/bfb0040758

[10] Simon Burton, Ibrahim Habli, Tom Lawton, John McDermid, Phillip Morgan, and Zoe Porter. 2020. Mind the gaps: Assuring the safety of autonomous systems from an engineering, ethical, and legal perspective. *Artificial Intelligence* 279 (Feb. 2020), 103201.

doi:10.1016/j.artint.2019.103201
[11] CARLA. 2025. *CARLA Autonomous Driving Leaderboard*. https://leaderboard.carla.org/leaderboard/
[12] Tsong Yueh Chen, Shing-Chi Cheung, and Siu-Ming Yiu. 1998. *Metamorphic Testing: A New Approach for Generating Next Test Cases*. Technical Report HKUST-CS98-01. Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong. https://www.cse.ust.hk/~scc/publ/CS98-01-metamorphictesting.pdf
[13] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *Comput. Surveys* 51, 1 (Jan. 2018), 1–27. doi:10.1145/3143561
[14] Mingfei Cheng, Yuan Zhou, and Xiaofei Xie. 2023. BehAVExplor: Behavior Diversity Guided Testing for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA '23)*. Association for Computing Machinery, New York, NY, USA, 488–500. doi:10.1145/3597926.3598072
[15] Mingfei Cheng, Yuan Zhou, Xiaofei Xie, Junjie Wang, Guozhu Meng, and Kairui Yang. 2024. Evaluating Decision Optimality of Autonomous Driving via Metamorphic Testing. arXiv:2402.18393 [cs.AI] https://arxiv.org/abs/2402.18393
[16] Tinkle Chugh, Karthik Sindhya, Jussi Hakanen, and Kaisa Miettinen. 2017. A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing* 23, 9 (Dec. 2017), 3137–3166. doi:10.1007/s00500-017-2965-0
[17] Victor Crespo-Rodriguez, Neelofar, and Aldeida Aleti. 2024. PAFOT: A Position-Based Approach for Finding Optimal Tests of Autonomous Vehicles. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test* (Lisbon, Portugal) *(AST '24)*. Association for Computing Machinery (ACM), New York, NY, USA, 159–170. doi:10.1145/3644032.3644457
[18] Kalyanmoy Deb and Ram Bhushan Agrawal. 1995. Simulated binary crossover for continuous search space. *Complex systems* 9, 2 (April 1995), 115–148.
[19] Yao Deng, Guannan Lou, Xi Zheng, Tianyi Zhang, Miryung Kim, Huai Liu, Chen Wang, and Tsong Yueh Chen. 2021. BMT: Behavior Driven Development-based Metamorphic Testing for Autonomous Driving Models. In *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET '21)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 32–36. doi:10.1109/met52542.2021.00012
[20] Yao Deng, Xi Zheng, Tianyi Zhang, Huai Liu, Guannan Lou, Miryung Kim, and Tsong Yueh Chen. 2023. A Declarative Metamorphic Testing Framework for Autonomous Driving. *IEEE Transactions on Software Engineering* 49, 4 (April 2023), 1964–1982. doi:10.1109/tse.2022.3206427
[21] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 78)*. PMLR, 1–16. https://proceedings.mlr.press/v78/dosovitskiy17a.html
[22] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 78)*, Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (Eds.). PMLR, 1–16. https://proceedings.mlr.press/v78/dosovitskiy17a.html
[23] Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. 2019. Compositional Falsification of Cyber-Physical Systems with Machine Learning Components. *Journal of Automated Reasoning* 63, 4 (Jan. 2019), 1031–1053. doi:10.1007/s10817-018-09509-5
[24] Manfred Eigen. 1971. Selforganization of matter and the evolution of biological macromolecules. *Die Naturwissenschaften* 58, 10 (Oct. 1971), 465–523. doi:10.1007/bf00623322
[25] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (Feb. 2013), 276–291. doi:10.1109/tse.2012.14
[26] Erik M. Fredericks, Mallory Jacobs, and Byron DeVries. 2024. Towards a Metamorphic Testing Architecture for Software-Defined Drone Systems. In *2024 11th International Conference on Software Defined Systems (SDS '24)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 170–177. doi:10.1109/sds64317.2024.10883896
[27] Jerome H. Friedman and Bogdan E. Popescu. 2008. Predictive learning via rule ensembles. *The Annals of Applied Statistics* 2, 3 (Sept. 2008), 916–954. doi:10.1214/07-aoas148
[28] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. 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) *(ISSTA '19, Vol. 4)*. Association for Computing Machinery, New York, NY, USA, 318–328. doi:10.1145/3293882.3330566
[29] David E. Goldberg and Jon Richardson. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, Vol. 4149. Hillsdale, NJ: Lawrence Erlbaum.
[30] Quentin Goss and Mustafa Ilhan Akbaş. 2022. Eagle Strategy with Local Search for Scenario Based Validation of Autonomous Vehicles. In *2022 International Conference on Connected Vehicle and Expo (ICCVE '22)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 1–6. doi:10.1109/iccve52871.2022.9743067
[31] Jia Cheng Han and Zhi Quan Zhou. 2020. Metamorphic Fuzz Testing of Autonomous Vehicles. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW '20)*. Association for Computing Machinery, New York, NY, USA, 380–385. doi:10.1145/3387940.3392252

[32] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 811–822. doi:10.1145/3510003.3510188

[33] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. 2021. Can Offline Testing of Deep Neural Networks Replace Their Online Testing?: A Case Study of Automated Driving Systems. *Empirical Software Engineering* 26, 90 (July 2021), 30 pages. doi:10.1007/s10664-021-09982-4

[34] Florian Hauer, Alexander Pretschner, and Bernd Holzmüller. 2019. Fitness functions for testing automated and autonomous driving systems. In *Computer Safety, Reliability, and Security (SAFECOMP '19)*. Springer International Publishing, Cham, 69–84. doi:10.1007/978-3-030-26601-1_5

[35] John H. Holland. 1992. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[36] Jeffrey Horn and David E. Goldberg. 1998. *A timing analysis of convergence to fitness sharing equilibrium*. Springer Science and Business Media LLC, Berlin, Heidelberg, 23–33. doi:10.1007/bfb0056846

[37] Dmytro Humeniuk, Foutse Khomh, and Giuliano Antoniol. 2023. AmbieGen: A search-based framework for autonomous systems testing. *Science of Computer Programming* 230 (Aug. 2023), 102990. doi:10.1016/j.scico.2023.102990

[38] Dmytro Humeniuk, Foutse Khomh, and Giuliano Antoniol. 2024. Reinforcement Learning Informed Evolutionary Search for Autonomous Systems Testing. *ACM Transactions on Software Engineering and Methodology* 33, 8 (Nov. 2024), 45 pages. doi:10.1145/3680468

[39] Rob J. Hyndman and Anne B. Koehler. 2006. Another look at measures of forecast accuracy. *International Journal of Forecasting* 22, 4 (Oct. 2006), 679–688. doi:10.1016/j.ijforecast.2006.03.001

[40] Muhammad Iqbal. 2024. *Metamorphic Relations for Testing Automated and Autonomous Driving Systems and Simulation Platforms*. Ph. D. Dissertation. School of Computing and Information Technology, University of Wollongong. https://ro.uow.edu.au/theses1/1758 Doctor of Philosophy thesis.

[41] Xinyu Ji, Lei Xue, Zhijian He, and Xiapu Luo. 2025. Autonomous Driving System Testing via Diversity-Oriented Driving Scenario Exploration. *ACM Transactions on Software Engineering and Methodology* 34, 7 (April 2025), 28 pages. doi:10.1145/3727875

[42] P. Kaelo and M.M. Ali. 2007. Integrated crossover rules in real coded genetic algorithms. *European Journal of Operational Research* 176, 1 (Jan. 2007), 60–76. doi:10.1016/j.ejor.2005.07.025

[43] Fauzia Khan, Mariana Falco, Hina Anwar, and Dietmar Pfahl. 2023. Safety Testing of Automated Driving Systems: A Literature Review. *IEEE Access* 11 (Oct. 2023), 120049–120072. doi:10.1109/access.2023.3327918

[44] Nicola Kolb, Florian Hauer, and Alexander Pretschner. 2021. Fitness Function Templates for Testing Automated and Autonomous Driving Systems in Intersection Scenarios. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC '21)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 217–222. doi:10.1109/itsc48978.2021.9564591

[45] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery (ACM), New York, NY, USA, 1675–1684. doi:10.1145/2939672.2939874

[46] Yuri Lavinas, Claus Aranha, Tetsuya Sakurai, and Marcelo Ladeira. 2018. Experimental Analysis of the Tournament Size on Genetic Algorithms. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC '18)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 3647–3653. doi:10.1109/smc.2018.00617

[47] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE '20)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 25–36. doi:10.1109/issre5003.2020.00012

[48] Rui Li, Huai Liu, Pak-Lok Poon, Dave Towey, Chang-Ai Sun, Zheng Zheng, Zhi Quan Zhou, and Tsong Yueh Chen. 2025. Metamorphic Relation Generation: State of the Art and Research Directions. *ACM Transactions on Software Engineering and Methodology* 34, 5 (May 2025), 25 pages. doi:10.1145/3708521

[49] Xuefang Li, Chengyuan Liu, Boli Chen, and Jingjing Jiang. 2022. Robust Adaptive Learning-Based Path Tracking Control of Autonomous Vehicles Under Uncertain Driving Environments. *IEEE Transactions on Intelligent Transportation Systems* 23, 11 (Nov. 2022), 20798–20809. doi:10.1109/tits.2022.3176970

[50] Christopher Lidbury, Andrei Lascu, Nathan Chong, and Alastair F. Donaldson. 2015. Many-core compiler fuzzing. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI '15)*. Association for Computing Machinery, New York, NY, USA, 65–76. doi:10.1145/2737924.2737986

[51] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* 22, 140 (1932), 55.

[52] Mikael Lindvall, Adam Porter, Gudjon Magnusson, and Christoph Schulze. 2017. Metamorphic Model-Based Testing of Autonomous Systems. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET '17)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 35–41. doi:10.1109/met.2017.6

[53] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE '22)*. Association for Computing Machinery, New York, NY, USA, 31–43. doi:10.1145/3540250.3549111

[54] Sean Luke. 2013. *Essentials of Metaheuristics* (second ed.). Lulu. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[55] Yixing Luo, Xiao-Yi Zhang, Paolo Arcaini, Zhi Jin, Haiyan Zhao, Fuyuki Ishikawa, Rongxin Wu, and Tao Xie. 2021. Targeting Requirements Violations of Autonomous Driving Systems by Dynamic Evolutionary Search. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 279–291. doi:10.1109/ase51524.2021.9678883

[56] Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. 2019. A Survey on Cooperative Co-Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 23, 3 (June 2019), 421–441. doi:10.1109/tevc.2018.2868770

[57] Samir W. Mahfoud. 1995. Population Size and Genetic Drift in Fitness Sharing. In *Foundations of Genetic Algorithms*. Vol. 3. Elsevier BV, 185–223. doi:10.1016/b978-1-55860-356-1.50014-5

[58] Claudio Mandrioli, Seung Yeob Shin, Domenico Bianculli, and Lionel Briand. 2025. Testing CPS with Design Assumptions-Based Metamorphic Relations and Genetic Programming. *IEEE Transactions on Software Engineering* 51, 6 (April 2025), 1–19. doi:10.1109/tse.2025.3563121

[59] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (March 1947), 50–60. doi:10.1214/aoms/1177730491

[60] Vincent Margot and George Luta. 2021. A New Method to Compare the Interpretability of Rule-Based Algorithms. *AI* 2, 4 (Nov. 2021), 621–635. doi:10.3390/ai2040037

[61] William M. McKeeman. 1998. Differential testing for software. *Digital Technical Journal* 10, 1 (1998), 100–107.

[62] Seyedali Mirjalili. 2018. *Genetic Algorithm*. Springer International Publishing, Cham, 43–55. doi:10.1007/978-3-319-93025-1_4

[63] Liviu Panait, Sean Luke, and Joseph F. Harrison. 2006. Archive-based cooperative coevolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (Seattle, Washington, USA) *(GECCO '06)*. Association for Computing Machinery, New York, NY, USA, 345–352. doi:10.1145/1143997.1144060

[64] Alain Petrowski. 1996. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation* (Nagoya, Japan) *(ICEC '96)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 798–803. doi:10.1109/icec.1996.542703

[65] Theofilos Petsios, Adrian Tang, Salvatore Stolfo, Angelos D. Keromytis, and Suman Jana. 2017. NEZHA: Efficient Domain-Independent Differential Testing. In *2017 IEEE Symposium on Security and Privacy (SP '17)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 615–632. doi:10.1109/sp.2017.27

[66] Mitchell A. Potter and Kenneth A. Jong. 1994. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature — PPSN III*. Springer Berlin Heidelberg, Berlin, Heidelberg, 249–257. doi:10.1007/3-540-58484-6_269

[67] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. 2020. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering* 25, 6 (Sept. 2020), 5193–5254. doi:10.1007/s10664-020-09881-0

[68] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Martins Mozeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. 2020. LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC '20)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 1–6. doi:10.1109/itsc45102.2020.9294422

[69] B. Sareni and L. Krahenbuhl. 1998. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation* 2, 3 (Sept. 1998), 97–106. doi:10.1109/4235.735432

[70] Luca Vittorio Sartori. 2019. Simulation-Based Testing to Improve Safety of Autonomous Robots. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW '19)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 104–107. doi:10.1109/issrew.2019.00053

[71] Sergio Segura, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. 2020. Metamorphic Testing: Testing the Untestable. *IEEE Software* 37, 3 (May 2020), 46–53. doi:10.1109/ms.2018.2875968

[72] S. Sette and L. Boullart. 2001. Genetic programming: principles and applications. *Engineering Applications of Artificial Intelligence* 14, 6 (Dec. 2001), 727–736. doi:10.1016/s0952-1976(02)00013-1

[73] Hao Shao, Letian Wang, Ruobing Chen, Hongsheng Li, and Yu Liu. 2023. Safety-Enhanced Autonomous Driving Using Interpretable Sensor Fusion Transformer. In *Proceedings of The 6th Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 205)*, Karen Liu, Dana Kulic, and Jeff Ichnowski (Eds.). PMLR, 726–737. https://proceedings.mlr.press/v205/shao23a.html

[74] Sepehr Sharifi, Donghwan Shin, Lionel C. Briand, and Nathan Aschbacher. 2023. Identifying the Hazard Boundary of ML-Enabled Autonomous Systems Using Cooperative Coevolutionary Search. *IEEE Transactions on Software Engineering* 49, 12 (Dec. 2023), 5120–5138. doi:10.1109/tse.2023.3327575

[75] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, and Yang Liu. 2023. A Survey on Automated Driving System Testing: Landscapes and Trends. *ACM Transactions on Software Engineering and Methodology* 32, 5 (July 2023), 1–62. doi:10.1145/3579642

[76] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 303–314. doi:10.1145/3180155.3180220

[77] Robert Tibshirani. 1996. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58, 1 (Jan. 1996), 267–288. doi:10.1111/j.2517-6161.1996.tb02080.x

[78] Handing Wang, Yaochu Jin, and Xin Yao. 2017. Diversity Assessment in Many-Objective Optimization. *IEEE Transactions on Cybernetics* 47, 6 (June 2017), 1510–1522. doi:10.1109/tcyb.2016.2550502

[79] Darrell Whitley. 2018. *Next Generation Genetic Algorithms: A User's Guide and Tutorial.* Springer International Publishing, Cham, 245–274. doi:10.1007/978-3-319-91086-4_8

[80] R. Paul Wiegand, William C. Liles, and Kenneth A. De Jong. 2001. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the genetic and evolutionary computation conference* (San Francisco) *(GECCO '01, Vol. 2611)*. Morgan Kaufmann, 1235–1245.

[81] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (Dec. 1945), 80–83. doi:10.2307/3001968

[82] Cort J. Willmott and Kenji Matsuura. 2005. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research* 30 (2005), 79–82. doi:10.3354/cr030079

[83] D. Randall Wilson and Tony R. Martinez. 1997. Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research* 6 (Jan. 1997), 1–34. doi:10.1613/jair.346

[84] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2024. *Experimentation in Software Engineering.* Springer Science and Business Media LLC, Berlin, Heidelberg. doi:10.1007/978-3-662-69306-3

[85] Lin Yang, Yanan Li, Deqing Huang, and Jingkang Xia. 2020. Iterative learning of an unknown road path through cooperative driving of vehicles. *IET Intelligent Transport Systems* 14 (March 2020), 423–431. Issue 5. doi:10.1049/iet-its.2019.0411

[86] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. 2011. Finding and understanding bugs in C compilers. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Jose, California, USA) *(PLDI '11)*. Association for Computing Machinery, New York, NY, USA, 283–294. doi:10.1145/1993316.1993532

[87] Zhenyu Yang, Ke Tang, and Xin Yao. 2008. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences* 178, 15 (Aug. 2008), 2985–2999. doi:10.1016/j.ins.2008.02.017

[88] Yao Yao. 2024. Enhancing autonomous driving systems with deep learning and spatial channel attention mechanisms: an experimental study. In *Fourth International Conference on Machine Learning and Computer Application (ICMLCA '23, Vol. 13176)*, Xinwei Yao and Xiangjie Kong (Eds.). International Society for Optics and Photonics, SPIE, 131761L. doi:10.1117/12.3029174

[89] Hossein Yousefizadeh, Shenghui Gu, Lionel C. Briand, and Ali Nasr. 2025. Using Cooperative Co-evolutionary Search to Generate Metamorphic Test Cases for Autonomous Driving Systems. *IEEE Transactions on Software Engineering* 51, 6 (2025), 1–30. doi:10.1109/tse.2025.3570897

[90] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE '18)*. Association for Computing Machinery, New York, NY, USA, 132–142. doi:10.1145/3238147.3238187

[91] Xiaokun Zheng, Huawei Liang, Biao Yu, Bichun Li, Shaobo Wang, and Zhilei Chen. 2020. Rapid Generation of Challenging Simulation Scenarios for Autonomous Vehicles Based on Adversarial Test. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA '20)*. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, NJ, USA, 1166–1172. doi:10.1109/icma49215.2020.9233535

[92] Zhi Quan Zhou and Liqun Sun. 2019. Metamorphic Testing of Driverless Cars. *Commun. ACM* 62, 3 (2019), 61–67.

A Experimental Results on GP_2

Based on GP_2, Fig. 13 compares the number of distinct solutions (*DS*) obtained by *CoCoMagic*, *SGA*, and *RS* over 10 runs. The x-axis shows the distance threshold θ_d, while the y-axis reports the average *DS*, with 95% confidence intervals as error bars. Results are shown across multiple fitness thresholds θ_f.

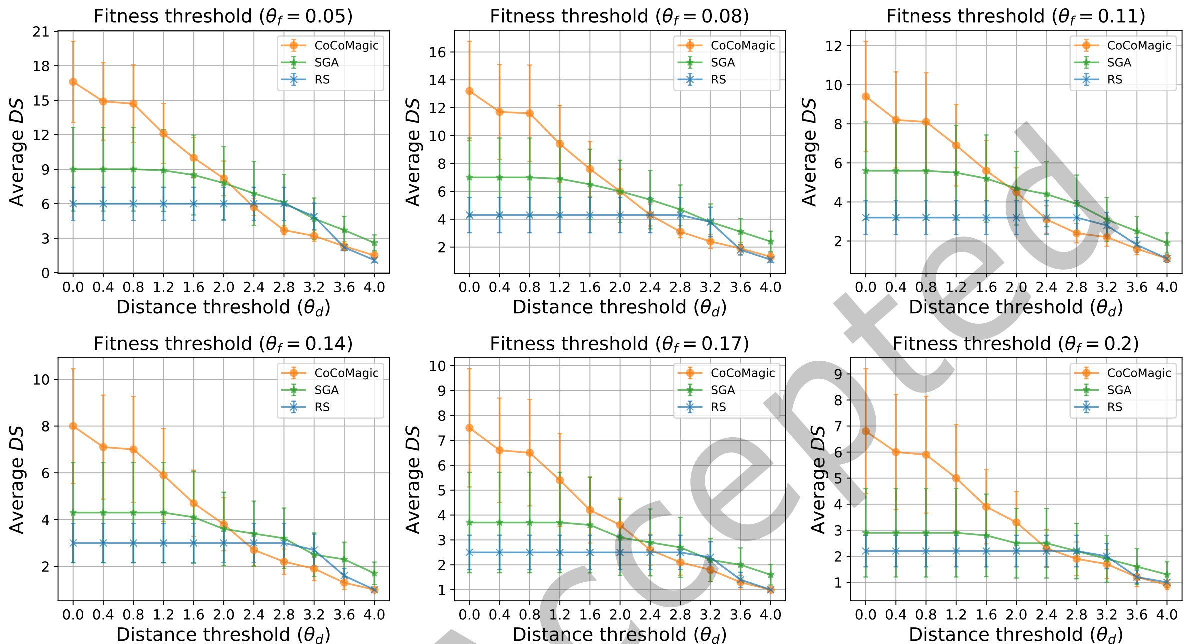

Fig. 13. *Distinct Solutions (DS)* vs. distance threshold (θ_d) across different fitness thresholds (θ_f). A higher *DS* value indicates that more distinct test cases have been discovered by the method. Each curve plots the average *DS* value at different θ_d settings, under a specific fitness threshold θ_f. This figure reveals how each method balances the quantity and diversity of test cases as θ_f varies.

Fig. 14 presents the distribution of fitness values obtained by *CoCoMagic*, *SGA*, and *RS* across 10 runs. Higher median values and upper ranges indicate a stronger ability to generate test cases that induce more severe behavioral differences between the two ADS versions.

Fig. 15 illustrates the progression of *DS* as the simulation budget increases for GP_2. Each subplot corresponds to a specific combination of θ_f and θ_d, with distinct curves for *CoCoMagic*, *SGA*, and *RS*. Similar to GP1, the plots suggest differences in efficiency as methods uncover severe and diverse behavioral discrepancies over time.

Finally, Fig. 16 presents the distribution of execution times (in hours) for *CoCoMagic*, *SGA*, and *RS* under a fixed search budget in GP_2. Similar to GP_1, each boxplot summarizes results across 10 runs, enabling a comparison of computational efficiency, where shorter times indicate faster completion.

Overall, the results for GP_2 exhibit similar trends to those observed for GP_1, reinforcing the consistency of our findings across different MR groups.

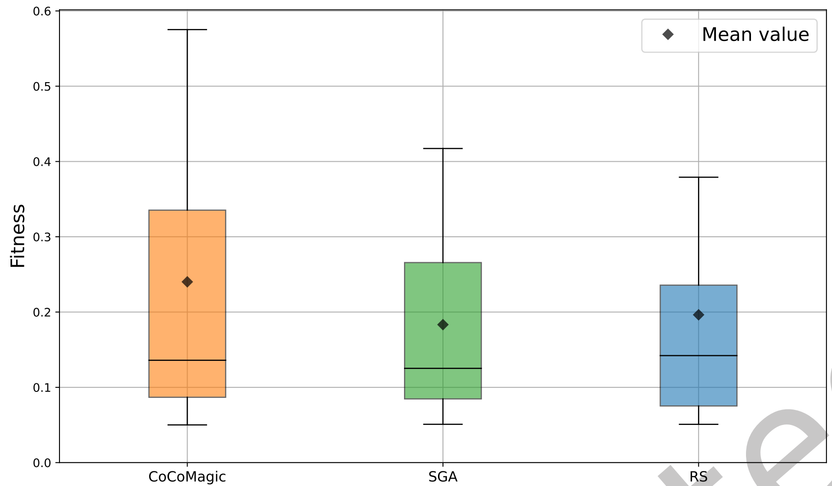

Fig. 14. Fitness distribution of test cases generated by *CoCoMagic*, *SGA*, and *RS*. Higher fitness indicates that the discovered test cases induce greater differences in the violations exhibited by the two ADSs. Each box shows the fitness values for the test cases across 10 executions, providing insight into each method's peak effectiveness. This figure highlights the relative strength of each method in uncovering critical test cases with severe behavioral differences.

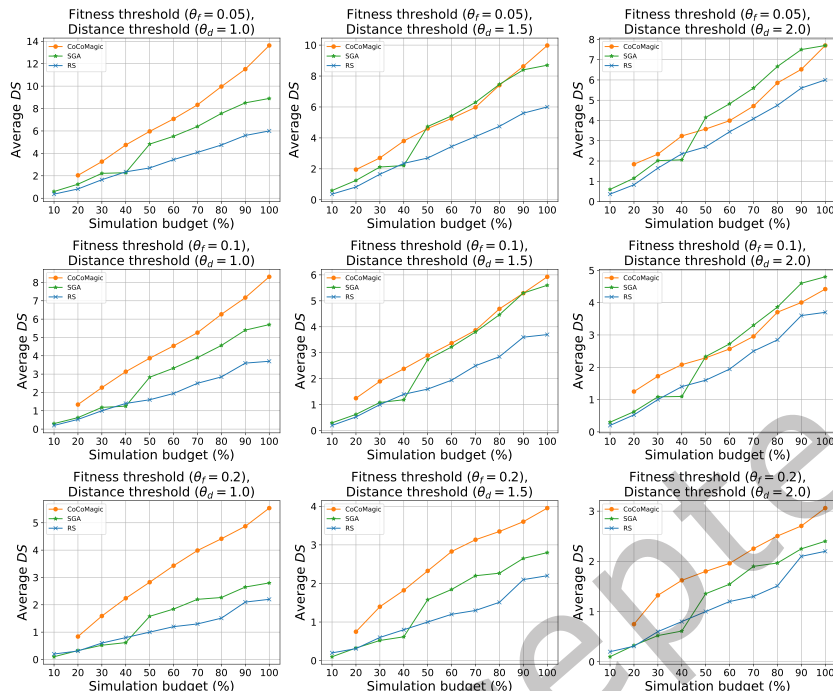

Fig. 15. *Distinct Solutions (DS)* progression across simulation budget levels. Each subplot illustrates how the *DS* value grows as more simulation resources are consumed, under specific combinations of fitness thresholds (θ_f) and distance thresholds (θ_d). The curves represent different methods, allowing a direct comparison of how quickly and effectively each approach uncovers severe and diverse behavioral discrepancies between ADS versions. Sharper increases and higher endpoints indicate greater efficiency in identifying test cases early in the search process.

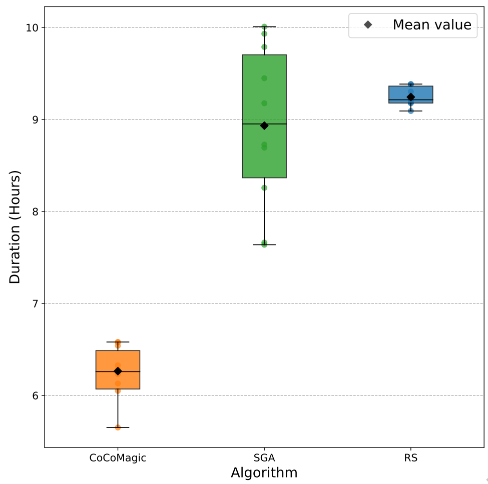

Fig. 16. Distribution of execution times (in hours) for *CoCoMagic*, *SGA*, and *RS* under a fixed search budget. Lower execution time indicates greater computational efficiency in completing the search. Each box summarizes the distribution of execution times across 10 runs.