Targeting Requirements Violations of Autonomous Driving Systems by Dynamic Evolutionary Search

Yixing Luo[*†], Xiao-Yi Zhang[‡], Paolo Arcaini[‡], Zhi Jin[*†], Haiyan Zhao[*†],
Fuyuki Ishikawa[‡], Rongxin Wu[§], Tao Xie[*†]
[*]Key Lab. of High-Confidence Software Technologies (Peking University), Ministry of Education, Beijing, China
[†]Department of Computer Science and Technology, School of EECS, Peking University, Beijing, China
[‡]National Institute of Informatics, Tokyo, Japan
[§]School of Informatics, Xiamen University, Xiamen, China
Email: {yixingluo, zhijin, zhhy.sei, taoxie}@pku.edu.cn {xiaoyi, arcaini, f-ishikawa}@nii.ac.jp
wurongxin@xmu.edu.cn

***Abstract*—Autonomous Driving Systems (ADSs) are complex systems that must satisfy multiple requirements such as safety, compliance to traffic rules, and comfortableness. However, satisfying all these requirements may not always be possible due to emerging environmental conditions. Therefore, the ADSs may have to make trade-offs among multiple requirements during the ongoing operation, resulting in one or more requirements violations. For ADS engineers, it is highly important to know which combinations of requirements violations may occur, as different combinations can expose different types of failures. However, there is currently no testing approach that can generate scenarios to expose different combinations of requirements violations. To address this issue, in this paper, we introduce the notion of *requirements violation pattern* to characterize a specific combination of requirements violations. Based on this notion, we propose a testing approach named EMOOD that can effectively generate test scenarios to expose as many requirements violation patterns as possible. EMOOD uses a prioritization technique to sort all possible patterns to search for, from the most to the least critical ones. Then, EMOOD iteratively includes an evolutionary many-objective optimization algorithm to find different combinations of requirements violations. In each iteration, the targeted pattern is determined by a dynamic prioritization technique to give preferences to those patterns with higher criticality and higher likelihood to occur. We apply EMOOD to an industrial ADS under two common traffic situations. Evaluation results show that EMOOD outperforms three baseline approaches in generating test scenarios by discovering more requirements violation patterns.**

***Index Terms*—Many-Objective Optimization, Autonomous Driving Systems, Requirements-Based Testing**

I. Introduction

Autonomous Driving Systems (ADSs) are making revolutionary changes in the domain of transportation. As ADSs are complex and safety-critical systems, their testing is vital for their wide acceptance [1]. Although on-road testing of autonomous vehicles is necessary and widely used in industrial ADSs such as Waymo [2] and Voyage.auto [3], existing studies [4], [5] have shown that billions of miles provide only limited assurance, since such time-consuming and costly testing approach would still miss dangerous and rare situations in the real world.

One-road testing for the ADS under test can be augmented by existing approaches [6]–[18] of virtual testing in computer simulations [18], [19] (in short as simulation-based testing), e.g., generating critical test scenarios in which the autonomous vehicle under test fails (e.g., the vehicle collides with obstacles). Considering that the test scenario space is complex and multidimensional, evolutionary search techniques [6]–[14] are applied to explore this test scenario space. In particular, these evolutionary search techniques aim to identify critical test scenarios that indicate complex driving conditions under which the ADS' behaviors violate specific requirements, e.g., safety [6], [7], [10] and compliance to traffic regulations such as lane keeping [9] and adhering to traffic light [8].

However, these existing simulation-based testing approaches do not consider combinations of requirements violations, which are highly important for two main reasons. First, satisfying all the requirements may not be possible for an ADS in practice, as unexpected events may happen in highly open and dynamic environments, e.g., intrusion of a hidden traffic participant or weather disturbances [20]. In response to these unexpected events, the control software of the ADS has to make trade-offs among requirements, likely resulting in one or more requirements being violated. Second, different combinations of requirements violations can expose different types of failures. For example, the type of failure in which the autonomous vehicle collides while running a red light is different from the one in which the autonomous vehicle collides while violating the lane keeping, as the different combinations of requirements violations may provide different insights about the cause of the collisions.

Generating scenarios that expose different combinations of requirements violations is a challenging problem. One idea is to enumerate all possible combinations of requirements violations as the objective functions in the evolutionary search,

This work is supported in part by the National Natural Science Foundation of China under Grant NO. 61620106007 and 61751210. Zhi Jin is the corresponding author. X. Zhang, P. Arcaini, and F. Ishikawa are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST, and MIRAI Engineerable AI Project (No. JPMJMI20B8), JST. We thank our industry partner Mazda for providing the software used in our work and discussing principles in testing and improving complex real-world automotive systems. The provided software is a prototype constructed for the purpose of evaluating new testing techniques, and its quality has no relation with the quality of Mazda products.

in order to identify critical test scenarios that expose different combinations of requirements violations. However, as the number of the requirements increases, the number of combinations of requirements violations grows exponentially; thus, testing all of them could be impossible under limited time and resources.

To address the preceding challenge, our work is based on an insight consisting of two aspects. First, the search should first focus on combinations of requirements violations with higher *criticality*, i.e., including more violated requirements with higher importance, as these combinations may lead to more serious accidents for the ADS under test. Second, some combinations of requirements violations are impossible to occur for the ADS in reality, and the search should not focus on them, in order to avoid wasting the search time.

Based on our insight, in this paper, we propose EMOOD, a search-based testing approach to generating diverse test scenarios that effectively expose different combinations of requirements violations in the ADS testing. Similar to existing work [6], [7], [10], we cast the problem of exposing different combinations of requirements violations into a search-based test-scenario generation problem. For EMOOD, we introduce the notion of *requirements violation pattern* (also referred to as pattern in the rest of the paper), i.e., a specific combination of requirements violations that the ADS under test can exhibit as the objective functions to search for. The notion of requirements violation pattern facilitates the determination of the ADS behaviors' violations of requirements. EMOOD conducts a type of *dynamic evolutionary search*, i.e., Evolutionary Many-Objective Optimization (EMOO), whose objectives are determined by Dynamic Prioritization (DP) that prioritizes the patterns in terms of criticality and likelihood to occur.

In particular, EMOOD starts with the initial ranking (of all the patterns) built statically based on the *criticality* of each requirement, and iteratively runs the heuristic dynamic prioritization algorithm (DP) to adjust the ranking additionally based on the dynamically estimated *likelihood of occurrence* of patterns: the more similar an unexposed violation pattern with exposed patterns is, the more likely the pattern can occur.

This paper makes the following main contributions:

- The notion of *requirements violation pattern* for characterizing different combinations of requirements violations.
- An effective approach for generating test scenarios to expose diverse requirements violation patterns, namely, EMOOD, which iteratively applies Evolutionary Many-Objective Optimization (EMOO) whose objective functions are selected by dynamic prioritization (DP).
- An initial prioritization technique (IP) for ranking patterns based on their criticality, i.e., the importance and number of violated requirements included in a pattern.
- A heuristic technique in DP for prioritizing patterns to dynamically rank them based on their criticality and likelihood of occurrence.
- Evaluation results for showing that EMOOD can discover diverse patterns with higher criticality, and given the same testing budget, it is more effective than baseline approaches that do not prioritize the patterns during the search.

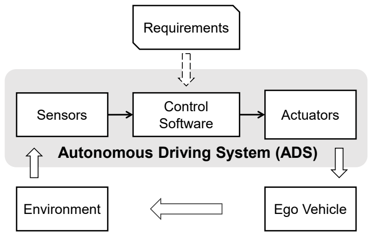
II. Motivation

Fig. 1(a) shows the decision process of an autonomous driving system (ADS), as reported in previous work [4]: the ADS is operating in a complex environment (including traffic participants). The autonomous vehicle running the ADS is called *ego vehicle*. The environment is perceived by sensors (e.g., camera, Lidar, and radar for object detection and localization). The ADS requirements are determined by different stakeholders, such as the passengers (e.g., comfortableness) and the authorities (e.g., safety standards [21]). Then, the ADS makes optimal decisions based on the observed situation and controls the vehicle through actuators, e.g., accelerating and direction changing. Both sensor inputs and actuator outputs are sequences of timestamped values. The ADS runs iteratively at regular time steps. At every time step, the ADS decides the optimal trajectory to be followed by minimizing cost functions related to different requirements to achieve (requirements violations are penalized in the cost function). In such a process, the ADS needs to make trade-offs among the requirements. The final output is the trajectory with the least overall cost (the red line in Fig. 1(b)).

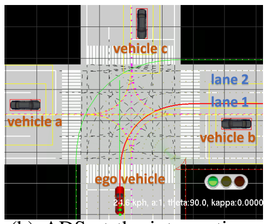
As a case study, we use an ADS with an optimization-based path planner (provided by our industry partner) called ADS_{PP}, which repeatedly uses a weighted cost function that considers various requirements to select the least costly path. ADS_{PP} can run in a simulator, as shown in Fig. 1(b). Besides the navigation mission, i.e., moving from an initial position to the destination, ADS_{PP} considers four categories of requirements during the planning process as follows:

- **Stability** (R_1): the ADS should assure the stable control and avoid dangerous actions for the vehicle [22]. $R_{1.1}$: the ADS should avoid impossible steering angles.
- **Safety** (R_2): the ADS should avoid collision with moving or static objects along the path [22]. $R_{2.1}$: the ADS should keep a safe distance from other objects.
- **Compliance** (R_3): the ADS should respect the traffic regulations enforced by law in a geographical area [22]. $R_{3.1}$: the velocity of the vehicle should be less than the speed limit; $R_{3.2}$: the vehicle should not run the red light; $R_{3.3}$: the vehicle should stay in the correct lane.
- **Comfortableness** (R_4): the planned trajectory should be comfortable for the passenger [23]. $R_{4.1}$: the vehicle's velocity should not change too much; $R_{4.2}$: the vehicle's acceleration should not change too much.

During the operation, ADS_{PP} is required to respond to the uncertain and dynamic environment, and adjust the trajectory for the *ego vehicle* to follow. For example, Fig. 1(b) shows a type of scenario in which the *ego vehicle* plans to turn right into *lane-1* at the intersection, while *vehicle-a* is crossing the intersection from left to right. However, there are different versions of this scenario that can lead to different behaviors of ADS_{PP}, and different satisfaction/violation of the requirements; we here consider two examples, i.e., *Scenario-A* and



(a) ADS decision process (adapted from [4])



(b) ADS at the intersection

Fig. 1: Overview of an autonomous driving system (ADS)

Scenario-B. In *Scenario-A*, *vehicle-a* proceeds at high speed, and the *ego vehicle* orders an emergency braking to bring the vehicle to a halt; however, the *ego vehicle* cannot stop in time and collides with *vehicle-a* (i.e., violation of requirement R_2). In *Scenario-B*, instead, the *vehicle-a* proceeds slightly slower than in *Scenario-A*, and, in this case, the *ego vehicle* accelerates and tries to cut into the lane before *vehicle-a*. This action plan requires a higher speed from the *ego vehicle*, to the point to exceed the speed limit; still, the *ego vehicle* cannot cut into the lane quickly enough, and collides with *vehicle-a* (i.e., it violates both R_2 and R_3). Although the *ego vehicle* collides in both scenarios, *Scenario-B* may be more interesting for debugging, because the occurrence of collision could be related to the violation of the speed limit requirement, which can provide a different insight about the cause of the collision. For example, if the *ego vehicle* decelerates to wait for *vehicle-a* to pass, the collision may not happen in *Scenario-B*.

Testing how ADS_{PP} handles the requirements requires finding critical scenarios in which one or more requirements are violated; such type of testing is particularly challenging. Furthermore, with the increase of the number of requirements, there could be different requirements violation combinations. We need approaches that can (1) find critical test scenarios for ADS_{PP} in which one or more requirements are violated; (2) expose all possible combinations of requirements violations for ADS_{PP}, if these combinations can occur in reality.

III. Formalization and Problem Statement

We here formalize an ADS and its environment by considering the input and configuration variables of ADS_{PP}.

A. Autonomous Driving System

At any time instant k, the state of an object is a tuple of three elements, i.e., $\boldsymbol{s}_k = (\boldsymbol{p}_k, \boldsymbol{v}_k, \boldsymbol{a}_k)$. The vector $\boldsymbol{p}_k = (x, y)$ is the geometric center of the position of the object, while $\boldsymbol{v}_k$ and $\boldsymbol{a}_k$ are its velocity and acceleration. The state of the ego vehicle is $\boldsymbol{s}^e_k$. We define $\mathcal{O}$ as the set of objects interacting with the ego vehicle, e.g., pedestrians, other vehicles; their state can be described as $\boldsymbol{s}^o_k$, with $o \in \mathcal{O}$. The trajectory of the ego vehicle is a sequence of the vehicle's states $\mathcal{T} = [\boldsymbol{s}^e_0, \ldots, \boldsymbol{s}^e_T]$, where the time interval between two consecutive states is fixed as τ, and T is the simulation time duration.

The ego vehicle continuously interacts with the dynamic and uncertain environments, and the ADS generates the trajectory for the ego vehicle to follow. A scenario Sce describes the environment in which the ego vehicle is operating, including (i) a map of the road structure; (ii) information of traffic regulations (e.g., location of traffic lights); (iii) dynamic behaviors of objects ($[\boldsymbol{s}^o_0, \ldots, \boldsymbol{s}^o_T], o \in \mathcal{O}$); (iv) initial state of the ego vehicle $\boldsymbol{s}^e_0$; (v) target destination for the ego vehicle $\boldsymbol{p}_d$; (vi) duration of the simulation T. We assume that the driving model of other vehicles is fixed and described by standard kinematic equations.

The system under test is the ADS of the ego vehicle. For simplicity, we view it as a function that, given a scenario Sce, produces the trajectory of the ego vehicle, i.e., $\mathcal{T}{=}ADS(Sce)$.

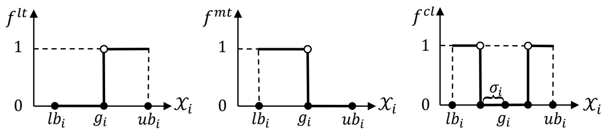


Fig. 2: Requirements violation evaluation functions

B. Requirements Violation Evaluation

It is challenging to verify the absolute violation/satisfaction (false/true) of requirements directly from the behaviors of the ADS operating in complex and changing environments [24]. In previous work [9], [10], various quantifiable metrics are provided to indicate the dangerous behaviors of the ADS. However, the previous work does not show how these metrics reflect requirements violation/satisfaction results.

To fill this gap, we provide a systematic way to design the quantifiable metrics (QMs) for requirements and their mapping functions to the requirements evaluation results. Given a requirements set $\mathcal{R} = \{R_1, \ldots, R_n\}$ for the ADS to consider during the decision process, we introduce $\mathcal{X}_i$ as the QM for each requirement R_i. $\mathcal{X}_i$ is defined as a function of the trajectory of the ego vehicle and the running scenario, i.e., $\mathcal{X}_i = h_i(\mathcal{T}, Sce)$. To evaluate the violation of requirement R_i, we compare $\mathcal{X}_i$ with the *threshold* g_i specified in the definition of R_i; in this way, we get the evaluation result $y_i = f(\mathcal{X}_i, R_i)$, ranging over the Boolean domain D_i. We set up three types of evaluation functions according to the relationship between the QM and the threshold in the requirement to indicate whether a requirement is violated ($y_i = 1$) or satisfied ($y_i = 0$). Fig. 2 shows the three Boolean evaluation functions, in which lb_i, ub_i, and g_i are the lower bound, upper bound, and threshold of $\mathcal{X}_i$, as specified in the requirements. The function $f^{lt}(\mathcal{X}_i)$ describes the LESS THAN relationship. A requirement R_i is satisfied if $\mathcal{X}_i \leq g_i$; otherwise, the requirement is violated and $y_i = 1$. Similarly, $f^{mt}(\mathcal{X}_i)$ describes the MORE THAN relationship ($g_i \leq \mathcal{X}_i$), and $f^{cl}(\mathcal{X}_i)$ describes the AS CLOSE AS POSSIBLE relationship ($g_i - \sigma_i \leq \mathcal{X}_i \leq g_i + \sigma_i$). Note that there can be other evaluation techniques to describe requirements satisfaction, such as fuzzy membership functions [25]. All the QMs of the requirements described in Section II can be assessed from our project website [26].

Example 1. Requirement $R_{2.1}$ states that the ego vehicle should keep safe distance from other objects. QM $\mathcal{X}_{2.1} = \min\limits_{k \in T, o \in \mathcal{O}} (\|\boldsymbol{p}_k^e - \boldsymbol{p}_k^o\|)$ is the minimum Euclidean distance between the ego vehicle and other objects including vehicles, pedestrians, etc. $\mathcal{X}_{2.1}$ should be MORE THAN the minimum separation ϵ_{min}; otherwise, there could be collisions. The evaluation function is formulated with function f^{mt} as follows:

$$y_{2.1} = \begin{cases} 0, & \epsilon_{min} \leq \mathcal{X}_{2.1} \\ 1, & otherwise \end{cases}$$

Definition 1 (Requirements Violation Pattern)**.** Let $\mathbb{D}$=$D_1 \times \ldots \times D_n$ be the space of evaluation results for all requirements, where D_j is the domain of y_j. A *requirements violation pattern* $V_i = [y_1, \ldots, y_n] \in \mathbb{D}$ is a vector, representing a distinct combination of requirements violation for the behavior of the ADS under test. Given the violation pattern V_i, the initial requirements set $\mathcal{R}$ can be split into two subsets of requirements, i.e., the set of satisfied requirements $\mathcal{R}_S^i = \{R_k | V_i[k] = 0\}$ and the set of violated requirements $\mathcal{R}_V^i = \{R_k | V_i[k] = 1\}$.

For example, $V_i = [1, 1, 0, 0, 0, 0, 0]$ represents a requirements violation pattern with the evaluation of seven requirements, where the first two are violated, and the other five are satisfied. The set of all possible requirements violation patterns is defined as $\mathcal{V} = \{V_0, \ldots, V_{m-1}\}$, where m is the total number of patterns in $\mathcal{V}$. For ADS_{PP}, we evaluate the requirements reported in Section II, and there are $2^7 = 128$ requirements violation patterns in $\mathcal{V}$.

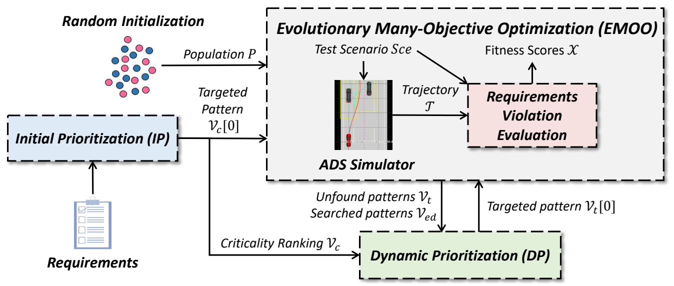
C. Problem Statement

For the ADS, the testing approach T targeting requirements violations can be defined as a function $\langle S, \mathcal{V}_f \rangle = T(ADS, \mathcal{R})$, where S is a set of critical test scenarios exposed by T and $\mathcal{V}_f$ is the set of requirements violation patterns that are covered by S ($\mathcal{V}_f \subseteq \mathcal{V}$). It is challenging for testers to explore the space of all possible scenarios to expose all the patterns with limited time or computing resources. Another challenge is that some patterns may not be achievable, and the testers should avoid wasting time searching for these patterns. In summary, the general problem that we aim to solve is as follows:
Given the ADS as the system under test, along with a set of requirements $\mathcal{R}$ that should be achieved by the system, design a testing approach T targeting requirements violations, which can effectively find test scenarios to expose different requirements violation patterns of high importance.

IV. Approach

A. Overview

Fig. 3 shows the workflow of EMOOD, which first applies Initial Prioritization (IP) and then iteratively applies Evolutionary Many-Objective Optimization (EMOO) and Dynamic Prioritization (DP) to achieve efficient ADS testing. IP is used for identifying the most critical requirements violation patterns. EMOO is used for finding scenarios violating and satisfying requirements, as specified by the given targeted pattern. This targeted pattern in a search round is identified by DP. In particular, DP aims to give preferences to those patterns with higher criticality and likelihood to occur. The combination of EMOO, IP, and DP makes it possible for our approach to explore different test scenarios, thereby exposing more types of requirements violation patterns.



Random Initialization
Evolutionary Many-Objective Optimization (EMOO)
Population P
Test Scenario Sce
Fitness Scores $\mathcal{X}$
Targeted Pattern $\mathcal{V}_c[0]$
Trajectory $\mathcal{T}$
Requirements Violation Evaluation
Initial Prioritization (IP)
ADS Simulator
Unfound patterns $\mathcal{V}_t$
Searched patterns $\mathcal{V}_{ed}$
Targeted pattern $\mathcal{V}_t[0]$
Criticality Ranking $\mathcal{V}_c$
Dynamic Prioritization (DP)
Requirements

Fig. 3: Overview of the proposed approach (EMOOD)

More specifically, the approach works as follows. In the beginning, in IP, given the ADS requirements and their importance, we rank the requirements violation patterns based on their criticality (see Section IV-B). The *criticality ranking* list of all patterns after IP is $\mathcal{V}_c$. Then, test generation starts by employing an *iterative* evolutionary process. At each iteration, the fitness functions of the searching process (i.e., EMOO) are defined based on the targeted pattern (see Section IV-C); in the first iteration, the targeted pattern is $\mathcal{V}_c[0]$. The population specifies values for the variables of an *abstract test scenario* (i.e., a scenario in which some fields are parameterized). It is instantiated by these values to produce a *concrete test scenario Sce* that is executed with the ADS simulator. The fitness scores $\mathcal{X}$ of the behaviors of ADS $\mathcal{T}$ in each test scenario are computed by running the ADS simulator and doing requirements violation evaluation.

After the execution of EMOO, we perform the dynamic prioritization (see Section IV-D). In particular, the pattern likelihood prioritization (see Alg. 1) is used to sort all possible requirements violation patterns based on their likelihood to occur. Such likelihood is estimated by considering the relation between exposed and unexposed patterns. The occurrence likelihood ranking of patterns $\mathcal{V}_l$ is merged with the criticality ranking $\mathcal{V}_c$ to update the list of patterns to search for $\mathcal{V}_t$, so as to identify the most critical patterns that are likely to occur. The first element $\mathcal{V}_t[0]$ of the merged list is used, in the next iteration, as the pattern targeted by EMOO.

Using DP, the approach continually changes its fitness functions round by round, until the given testing time or resource budget exhausts. The whole process is described in Section IV-E in detail. The output is a set of test scenarios that facilitate reproducing the wrong behaviors of the ADS and the set of requirements violation patterns that have been identified during the testing.

B. Initial Prioritization (IP)

As it is time-consuming and economically expensive to test whether each pattern in the set $\mathcal{V}$ is possible to occur, we need to determine their order of being searched for. In the beginning, we prioritize the patterns based on their *criticality*, which is defined based on the following two ranking rules:

- $\mathcal{P}_1$: a pattern V_i is ranked higher than another pattern V_j if the highest *importance level* of the violated requirements in V_i is higher than that in V_j.
- $\mathcal{P}_2$: a pattern V_i is ranked higher than another pattern V_j if V_i has more violated requirements in importance level p than V_j, while V_i has the same number of violated requirements in importance level l as V_j for $\forall p < l <= q$ where q is the maximum importance level.

Rule $\mathcal{P}_1$ is designed based on the predefined importance levels among different categories of requirements in ADS behaviors as defined in previous work [22] (e.g., the safety requirement's importance level is higher than that of the compliance requirement). For the functionality of the system, certain requirement R_i may be split into a set of sub-requirements $\{R_{i.1}, ..., R_{i.M_i}\}$ to be achieved, where M_i is the number of sub-requirements (e.g., R_3 is instantiated into $\{R_{3.1}, R_{3.2}, R_{3.3}\}$). For simplicity, we assign the same importance level to the sub-requirements belonging to the same category, as done in ADS_{PP} provided by our industry partner. For ADS_{PP}, the requirements listed in Section II rank from the highest to the lowest one, i.e., $R_1 \succ R_2 \succ R_3\{R_{3.1}, R_{3.2}, R_{3.3}\} \succ R_4\{R_{4.1}, R_{4.2}\}$. Thus, according to $\mathcal{P}_1$, V_i is ranked higher than ($\succ_{\mathcal{P}_1}$) V_j if $\exists R_k \in \mathcal{R}_V^i, \forall R_{k'} \in \mathcal{R}_V^j : R_k \succ R_{k'}$.

Assume that importance level p is the highest level where patterns V_i and V_j differ in terms of the number of violated requirements in the same level. Rule $\mathcal{P}_2$ is designed based on the assumption that in the critical scenarios in which the trajectory generated by the ADS violates a larger number of requirements in importance level p, the wrong decision logic of the control software is more likely to be exposed.

We apply these two ranking rules, assuming that $\mathcal{P}_1$ takes precedence over $\mathcal{P}_2$, as suggested by our industry partner. Based on $\mathcal{P}_1$ and $\mathcal{P}_2$, to compare the criticality of patterns V_i and V_j is to compare the number of violated requirements level by level from the highest to the lowest importance levels. Therefore, the set of all requirements violation patterns $\mathcal{V}$ can be transformed into a sorted list of patterns $\mathcal{V}_c$=$\mathcal{P}_{\{1,2\}}(\mathcal{V})$, ranging from the most to the least critical patterns; we name the list *criticality ranking*. Note that $\mathcal{V}_c$ is a partial order, and some patterns could have the same ranking. The first element $\mathcal{V}_c[0]$ in $\mathcal{V}$ is a unit vector where all the requirements are violated, while the last element $\mathcal{V}_c[m-1]$ is the zero vector indicating that all requirements are satisfied.

C. Evolutionary Many-Objective Optimization (EMOO)

As it is difficult to generate critical test scenarios in which one or more requirements are violated [27], we use a search-based testing (SBT) approach, which has been shown to be very effective for ADS testing [6], [9], [12]. We cast the problem of generating ADS critical test scenarios for a specific requirements violation pattern as a many-objective optimization problem [28], where the fitness functions are defined as the indicators for requirements satisfaction/violation.

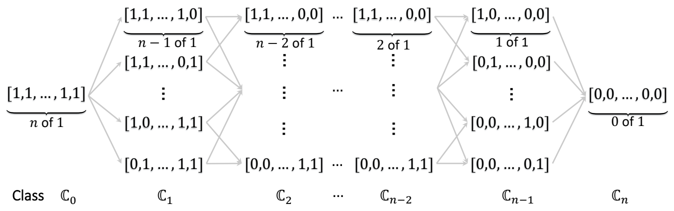


Fig. 4: The relationship of requirements violation patterns

For the many-objective optimization algorithm, we use the Non-dominated Sorting Genetic Algorithm version 3 (NSGA-III) [29], as it is good in solving problems with many objectives (four or more) [30] and so is suitable for our case. In our work, NSGA-III generates a number of ADS critical test scenarios by maximizing or minimizing the indicators that characterize the violation or satisfaction of the requirements (i.e., QMs $\mathcal{X}_i$ introduced in Section III-B). Here, we discuss how we apply NSGA-III to our case.

Individuals. As explained in Section III-A, test scenarios describe the characteristics of the operating environment and the initial state of the ego vehicle. In EMOO, we aim at generating test scenarios. However, searching over all the possible scenarios is not feasible. Hence, we follow a pragmatic approach also adopted by other work [31], [32]. Specifically, we define an *abstract scenario* that specifies some fixed characteristics, such as the road structure and the other traffic participants; in the abstract scenario, some characteristics (e.g., initial position and acceleration of a vehicle) are parameterized with variables defined over some domains. In the search, an individual is an assignment to these variables. Given an individual, a *concrete test scenario* can be derived by instantiating the abstract scenario with the values assigned to the variables in the individual.

Fitness Functions. For a targeted requirements violation pattern $V_i \in \mathcal{V}$, there is a corresponding set of quantitative fitness functions $\mathcal{F} = \{F_1, \ldots, F_n\}$, whose values indicate the QMs of the requirements. Since the requirements evaluation is Boolean as shown in Fig. 2, V_i is an n-dimensional vector of $\{0, 1\}$. For $V_i[k]$=0, the satisfaction of the R_k should be guaranteed ($R_k \in \mathcal{R}_S^i$), while for $V_i[k] = 1$, R_k is expected to be violated ($R_k \in \mathcal{R}_V^i$). Based on the set of satisfied and violated requirements $\mathcal{R}_S^i$ and $\mathcal{R}_V^i$, the set of fitness functions $\mathcal{F}$ is also split into $\mathcal{F}_S^i$ and $\mathcal{F}_V^i$. For $R_k \in \mathcal{R}_S^i$, the corresponding fitness function $F_k \in \mathcal{F}_S^i$ is to improve $\mathcal{X}_k$, i.e., trying to satisfy R_k. For $R_k \in \mathcal{R}_V^i$, its corresponding fitness function $F_k \in \mathcal{F}_V^i$ is to worsen $\mathcal{X}_k$, i.e., trying to violate R_k.

D. Dynamic Prioritization (DP)

In this phase, we select a pattern to be used as the target for the next search round of EMOO. The rationale is that we want to select a critical pattern that can occur in reality. Thus, DP first computes a ranking of patterns based on their likelihood of occurrence. Then, DP merges this ranking with the ranking based on the criticality $\mathcal{V}_c$.

Algorithm 1: Pattern Likelihood Prioritization

```
Input: $\mathcal{V}_t$: patterns that have not been found;
       $\mathcal{V}_{ed}$: patterns that have been searched for.
Output: $\mathcal{V}_l$: likelihood rankings for patterns
1  Initialize the reward of each pattern with Eq. (1);
2  $\mathcal{V}_l \leftarrow \mathcal{V}_t \setminus \mathcal{V}_{ed}$;
3  for $V_j \in \mathcal{V}_l$ do
4      $k' \leftarrow GetClass(V_j)$ ;
5      for $V_i \in (\mathcal{V} \setminus \mathcal{V}_t) \cup (\mathcal{V}_{ed} \cap \mathcal{V}_t)$ do
6          if $V_i$ is predecessor to $V_j$ then
7              $k \leftarrow GetClass(V_i)$;
8              $r(V_j) \mathrel{+}= \gamma^{k'-k} \cdot r(V_i)$
9  $\mathcal{V}_l \leftarrow sort(\mathcal{V}_l, r(\mathcal{V}_l))$;
10 return $\mathcal{V}_l$
```

1) Likelihood Prioritization: The intuition of *occurrence likelihood ranking* for requirements violation patterns is that if a specific requirements violation pattern has been found, there is a high likelihood to expose similar/related violation patterns that are less critical than the found pattern. To prioritize the patterns, we first define the relationship between different patterns to estimate the likelihood of their occurrence.

Definition 2. (Requirement Violation Pattern Relationship) Given two requirements violation patterns $V_k = [y_1, \ldots, y_n]$ and $V_{k'} = [y'_1, \ldots, y'_n]$, if $\exists i \in \{1, \ldots, n\}\colon y_i = 1 \wedge y'_i = 0 \wedge (\forall j \in \{1, \ldots, n\} \setminus \{i\}\colon y_j = y'_j)$, then pattern V_k is the *predecessor* of pattern $V_{k'}$, and $V_{k'}$ is the *successor* of V_k.

As shown in Fig. 4, we construct a graph to describe the relationship between different requirements violation patterns based on Def. 2. The relationship between two patterns is described with an arrow ($\rightarrow$), where the left-hand side of the arrow indicates the direct predecessor and the right-hand side of the arrow indicates the successor. Note that a requirements violation pattern V_i could have multiple direct predecessors, as well as multiple successors. Therefore, the set of patterns $\mathcal{V}$ is a partially ordered set according to the relationship. The pattern with all requirements violated ($[1, 1, \ldots, 1, 1]$), has no predecessors, while the pattern with no requirements violated ($[0, 0, \ldots, 0, 0]$) has no successors. All the patterns can be classified into $n+1$ classes ($\mathbb{C}_0, \ldots, \mathbb{C}_n$) based on the number of violated requirements.

Alg. 1 shows the prioritization based on likelihood that exploits the pattern relationship introduced in Def. 2. First, we initialize a reward function for each pattern V_i as $r(V_i)$ (Line 1), indicating the likelihood that the pattern occurs in reality. The higher the reward is, the higher the likelihood of the pattern is to be found. Given the list $\mathcal{V}_t$ of patterns that have not been found so far, and the list $\mathcal{V}_{ed}$ of patterns that have been treated as fitness functions to search for, the reward for each requirement violation pattern is initialized as follows:

$$r(V_i) = \begin{cases} r_0^+, & V_i \in \mathcal{V} \setminus \mathcal{V}_t \\ r_0^-, & V_i \in \mathcal{V}_{ed} \cap \mathcal{V}_t \\ 0, & \mathcal{V}_t \setminus \mathcal{V}_{ed} \end{cases} \tag{1}$$

For requirements violation patterns that have been found ($\mathcal{V} \setminus \mathcal{V}_t$), we assign a positive reward r_0^+, and for patterns that have been searched but not found (i.e., $\mathcal{V}_{ed} \cap \mathcal{V}_t$), we define a constant penalty as r_0^-. The remaining patterns are those that need to be searched for (i.e., $\mathcal{V}_t \setminus \mathcal{V}_{ed}$), and their reward is initialized as 0; we identify them as $\mathcal{V}_l$ (Line 2). The rewards of patterns in $\mathcal{V}_l$ are updated as described as follows.

To calculate the rewards of patterns $\mathcal{V}_l$ that we need to search for, we consider the rewards of their predecessors. The idea is that if a predecessor V_i of a requirements violation pattern V_j has been previously found ($V_i \in \mathcal{V} \setminus \mathcal{V}_t$), it is likely that V_j may occur. On the other hand, if V_i has been searched for but not found ($V_i \in \mathcal{V}_{ed} \cap \mathcal{V}_t$), the likelihood that pattern V_j can be found is low. Based on these heuristic rules, the algorithm works as follows. Given a pattern V_j that has not been searched for and not been found (Line 3), the algorithm iterates over the other patterns V_i initialized with non-zero rewards (Line 5); for those that are predecessors of V_j (Line 6), it updates the reward of V_j with reward $r(V_i)$ discounted based on the distance between V_i and V_j (Line 8). Note that $\gamma \in (0, 1)$, and so the factor $\gamma^{k'-k}$ decreases as the distance increases. Finally, it prioritizes the sequence $\mathcal{V}_l$ of requirements violation patterns to search for, based on the computed rewards ranging from the highest to the lowest (Line 9), as the higher the reward, the higher the likelihood that this pattern is exposed in the next round. We call $\mathcal{V}_l$ as *(estimated) likelihood ranking*.

2) Merging Criticality and Likelihood Rankings: Then, we merge the rankings of requirements violation patterns in $\mathcal{V}_l$ and $\mathcal{V}_c$ with a weighted sum, following a classical approach used in multi-criteria decision-making [33]. The *criticality ranking* $\mathcal{V}_c$ indicates the importance of the patterns, while the *likelihood ranking* $\mathcal{V}_l$ indicates the likelihood of the occurrence of each pattern. By combining these two ranking algorithms, we can select (for the next round of EMOO) the patterns that are both critical and likely to be found. The weights for these two lists are defined as a vector $[w_c, w_l]$, and the final rank of a pattern V_i is determined by $\sum_{j \in \{c,l\}} w_j \cdot rank(\mathcal{V}_j(V_i))$, where $rank(\mathcal{V}_j(V_i))$ is the function to retrieve the rank of V_i in $\mathcal{V}_j$. The merged ranking substitutes $\mathcal{V}_t$.

E. Details of EMOOD

We here describe the details of our `EMOOD` approach (as shown in Fig. 3), i.e., how to combine the initial prioritization (IP, Section IV-B), the evolutionary many-objective optimization (EMOO, Section IV-C), and the dynamic prioritization (DP, Section IV-D) during the testing process. Alg. 2 shows the algorithm of `EMOOD`, aiming at generating test scenarios that expose different requirements violation patterns.

`EMOOD` receives a set $\mathcal{V}$ of requirements violation patterns to search for, the total number M of generations, and the budget G of generations in each search round. The output of `EMOOD` is a set S of test scenarios and requirements violation patterns $\mathcal{V}_f$ that have been found.

Initially, the approach applies initial prioritization (see Section IV-B) to rank the patterns by criticality (Line 1), obtaining ranking $\mathcal{V}_c$. The list of patterns to search for $\mathcal{V}_t$ is initialized with $\mathcal{V}_c$ (Line 2). Then, at each iteration, EMOO

Algorithm 2: EMOOD

```
Input: $\mathcal{V}$: a set of requirements violation patterns;
       $M$: total number of generations;
       $G$: budget of generations for each execution of EMOO.
Output: $S$: a set of test scenarios;
        $\mathcal{V}_f$: requirements violation patterns that have been found
1  $\mathcal{V}_c \leftarrow \mathcal{P}_{\{1,2\}}(\mathcal{V})$;                          // IP
2  $\mathcal{V}_t \leftarrow \mathcal{V}_c, \mathcal{V}_{ed} \leftarrow \emptyset$;
3  $Pop \leftarrow \emptyset$;
4  while $M > 0 \land \mathcal{V}_t \neq \emptyset$ do
5      Select an initial population set $P$ randomly;
6      $\mathcal{F} \leftarrow ObjGenerate(\mathcal{V}_t[0])$;
7      $Q, g \leftarrow$ NSGA-III$(G, P, \mathcal{F}, stop\_condition)$;        // EMOO
       // g: used generations; Q: all solutions
8      $M = M - g$;
9      $Q \leftarrow Q \cup Pop$ ;
10     $\mathcal{V}_{ed} \leftarrow \mathcal{V}_{ed} \cup \{\mathcal{V}_t[0]\}$;
11     $\mathcal{V}_t \leftarrow GetUnfoundPatterns(Q, \mathcal{V}_t)$;
12     $\mathcal{V}_l \leftarrow RelationRanking(\mathcal{V}_t, \mathcal{V}_{ed})$;                 // DP
13     $\mathcal{V}_t \leftarrow MergedRanking(\mathcal{V}_l, \mathcal{V}_c)$;                  // DP
14 $S \leftarrow SceneEncoding(Pop), \mathcal{V}_f \leftarrow \mathcal{V}_c \setminus \mathcal{V}_t$;
15 return $\langle S, \mathcal{V}_f \rangle$
```

starts with a randomly selected population set P (Line 5), and a requirement violation pattern to search for that defines the fitness functions (Line 6): the pattern is the first element of the list of unfound patterns $\mathcal{V}_t$. EMOO performs a number g of generations using NSGA-III [29] (Line 7) until the $stop\_condition$ is reached: either the total number G of generations is reached or the targeted pattern has been found. The number g of used generations is subtracted from the total budget M of generations (Line 8). The pattern that has been searched for is stored in set $\mathcal{V}_{ed}$ (Line 10). Then, $\mathcal{V}_t$ is updated by removing the patterns that have been found (the targeted one and/or also other patterns) according to the requirements violation evaluations of all solutions generated during search (Line 11). Note that some of the patterns, despite having been searched for, may not be discovered within their dedicated search round, either because they are too difficult to discover or they are not achievable at all.

Finally, EMOOD applies dynamic prioritization: it first uses Alg. 1 to get the likelihood ranking $\mathcal{V}_l$ of the patterns (Line 12); then, it merges the rankings $\mathcal{V}_t$ and $\mathcal{V}_l$ with a weighted sum (Line 13). EMOOD can be stopped when the total number M of generations is reached or all patterns have been found, i.e., $\mathcal{V}_t \neq \emptyset$ (Line 4). Note that, in the ADS testing, the most time-consuming part of the search is running simulations to compute fitness functions. EMOOD does not increase the number of simulations compared to EMOO with the original NSGA-III.

V. Evaluation

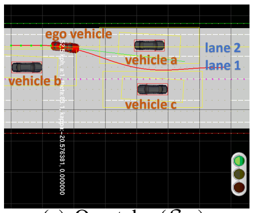
To demonstrate the effectiveness of our approach, we investigate the following research questions.

RQ1 (Pattern Detection): How is the exploration capability of EMOOD in generating test scenarios to expose different requirements violation patterns?

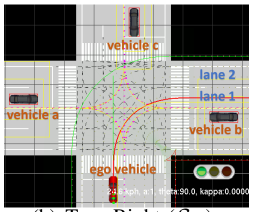
RQ2 (Pattern Criticality): How effective is EMOOD in discovering critical requirements violation patterns?

TABLE I: Two traffic scenarios used in the experiments

Name	Description	#Vars.
Overtake ($\mathcal{S}_O$)	The ego vehicle tries to overtake another *vehicle-a* proceeding slowly, while *vehicle-b* is coming from behind.	15
Turn Right ($\mathcal{S}_T$)	The ego vehicle must turn right at the intersection. Another *vehicle-a* is crossing from left, and *vehicle-b* from right. *Vehicle-c* is waiting at the intersection.	12



(a) Overtake ($\mathcal{S}_O$)



(b) Turn Right ($\mathcal{S}_T$)

Fig. 5: Two abstract traffic scenarios for the experiments

A. Experimental Design and Settings

1) Traffic Scenarios: We consider two abstract traffic scenarios (where left-hand traffic is assumed), as shown in Table I. Each abstract traffic scenario defines shared characteristics of the generated test scenarios (e.g., road map, speed limit), and specific characteristics (e.g., position, speed, acceleration of the ego vehicle and other vehicles, duration of the traffic light) that are left as search variables for EMOO. The ego vehicle with ADS_{PP} follows the requirements reported in Section II.

The two abstract traffic scenarios identify two different driving situations that will possibly lead to different behaviors of ADS_{PP}, and thus patterns of potential requirements violations. To ensure that the simulations start from a valid and meaningful state, we have the following two constraints on the initial state of the simulation (also used for other ADSs in previous work [8]): (1) there is a safe distance between the ego vehicle and other vehicles when all the vehicles start, and (2) the ego vehicle is far from the traffic light such that it has sufficient time to react.

Overtake ($\mathcal{S}_O$): As shown in Fig. 5(a), the ego vehicle is proceeding on *lane-2*, encounters *vehicle-a* proceeding slowly, and tries to overtake it. Meanwhile, *vehicle-b* is coming from behind the ego vehicle on the passing lane *lane-1*. Moreover, *vehicle-c* is proceeding in the opposite direction on a different lane. In $\mathcal{S}_O$, there are 15 search variables, including the initial states (i.e., position, velocity, and acceleration) of the ego vehicle, *vehicle-a*, *vehicle-b*, and *vehicle-c*, respectively, and the location and duration of the traffic light. Table II reports the search space for the traffic scenario $\mathcal{S}_O$.

Turn Right ($\mathcal{S}_T$): As shown in Fig. 5(b), the ego vehicle turns right at the intersection. Meanwhile, *vehicle-a* is crossing from left, *vehicle-b* is crossing from right, and *vehicle-c* is waiting at the intersection. As shown in Table II, there are 12 search variables: the initial states of the ego vehicle and the other three vehicles, respectively, and the duration of the traffic light.

TABLE II: Search spaces for traffic scenarios $\mathcal{S}_O$ and $\mathcal{S}_T$

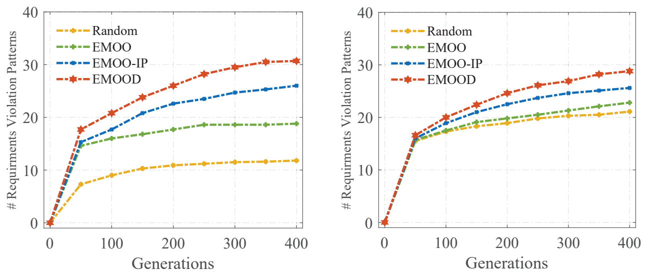
Objects		Variables	Intervals
$\mathcal{S}_O$	*ego vehicle*	$\boldsymbol{p}_0^e[y]$, $\boldsymbol{v}_0^e$	[5, 25], [5, 16]
	vehicle-a	$\boldsymbol{p}_0^a[y]$, $\boldsymbol{v}_0^a$, $\boldsymbol{a}_0^a$	[45, 65], [4, 16], [0, 3]
	vehicle-b	$\boldsymbol{p}_0^b[y]$, $\boldsymbol{v}_0^b$, $\boldsymbol{a}_0^b$	[0, 20], [4, 16], [0, 3]
	vehicle-c	$\boldsymbol{p}_0^c[y]$, $\boldsymbol{v}_0^c$, $\boldsymbol{a}_0^c$	[40, 90], [4, 16], [0, 3]
	traffic light	$\boldsymbol{p}[y], t_g, t_y, t_r$	[70, 80], [5, 10], [1, 2], [2, 4]
$\mathcal{S}_T$	*ego vehicle*	$\boldsymbol{p}_0^e[y]$, $\boldsymbol{v}_0^e$	[150, 190], [4, 16]
	vehicle-a	$\boldsymbol{p}_0^a[x]$, $\boldsymbol{v}_0^a$, $\boldsymbol{a}_0^a$	[-75, -25], [4, 16], [0, 3]
	vehicle-b	$\boldsymbol{p}_0^b[x]$, $\boldsymbol{v}_0^b$, $\boldsymbol{a}_0^b$	[25, 75], [4, 16], [0, 3]
	vehicle-c	$\boldsymbol{p}_0^c[y]$	[225, 235]
	traffic light	t_g, t_y, t_r	[6, 12], [1, 2], [2, 4]

2) Baseline Approaches: To the best of our knowledge, at the moment of writing, there do not exist any automated testing approaches targeting requirements violation patterns. We compare `EMOOD` with three approaches, including the random test generation algorithm (the baseline of comparison typically adopted in SBSE research [34]) and two variants of `EMOOD`, to demonstrate the effectiveness of our approach in terms of pattern detection and coverage.

- `Random`: it randomly generates values for variables in the search space of $\mathcal{S}_O$ and $\mathcal{S}_T$ to create test scenarios and checks which patterns are discovered.
- `EMOO`: application of EMOO using fitness functions targeting pattern $\mathcal{V}_c[0]$ (i.e., all requirements violated). Fitness functions are never changed over the search process. Note that, since $\mathcal{V}_c[0]$ is the most critical pattern and the ancestor of all the patterns, other patterns can be discovered while trying to cover it.
- `EMOO-IP`: the combination of initial prioritization with updated objective functions but without the dynamic prioritization. In this case, each EMOO execution uses the most critical pattern (in $\mathcal{V}_c$) that has not been searched so far.

3) Configurations and Implementations: We implement the baselines and our approach in Python, and use jMetalPy [35] as the search framework. For `Random`, we use the implementation of random search in jMetalPy [35]. For the settings of the EMOO algorithm, we adopt NSGA-III [29]. The settings for NSGA-III are the default ones in jMetalPy: parent selection with tournament selection [36], SBX crossover operator, polynomial mutation operator, crossover rate of 100%, and mutation rate equal to the reciprocal of the number of search variables. For each EMOO-based algorithm, the size of the population is 50. The weights used when merging $\mathcal{V}_c$ and $\mathcal{V}_l$ are $[w_c, w_l] = [1, 1]$.

Each approach (`EMOOD` and the baselines) is used to test the ADS for some generations. As the termination condition G for `EMOO`, `EMOO-IP`, and `EMOOD`, we set 100 generations for the first round and 50 generations for the subsequent rounds. As global budget M, we set a total number of 400 generations across all the rounds. Hence, totally we can evaluate $50 \times 400 = 20000$ scenarios. For `Random`, in order to assure a fair comparison, we set 20000 fitness evaluations as the termination condition. To take into account the random effect during the search, we repeat each algorithm ten times.



(a) Overtake ($\mathcal{S}_O$). (b) Turn Right ($\mathcal{S}_T$).

Fig. 6: RQ1 – Average number of discovered patterns

TABLE III: RQ1 – Statistical test results comparing the number of patterns discovered by the approaches

Approaches	Overtake ($\mathcal{S}_O$)		Turn Right ($\mathcal{S}_T$)	
	p-value	$\hat{A}_{12}$	p-value	$\hat{A}_{12}$
`EMOOD` vs. `Random`	1.64e-4	1.00	1.22e-4	1.00
`EMOOD` vs. `EMOO`	1.73e-4	1.00	1.57e-4	1.00
`EMOOD` vs. `EMOO-IP`	1.58e-4	1.00	2.38e-4	0.99
`EMOO-IP` vs. `Random`	1.54e-4	1.00	1.19e-4	1.00
`EMOO-IP` vs. `EMOO`	1.62e-4	1.00	8.06e-4	0.94
`EMOO` vs. `Random`	6.06e-4	0.96	0.0012	0.92

The experiments are executed on servers with CPU (Intel Xeon E5-2697A V4@2.6GHz), 32 cores, and 128 GB of memory. The time budget for scenario simulation is set as 100 seconds. More details of experimental results can be found online [26].

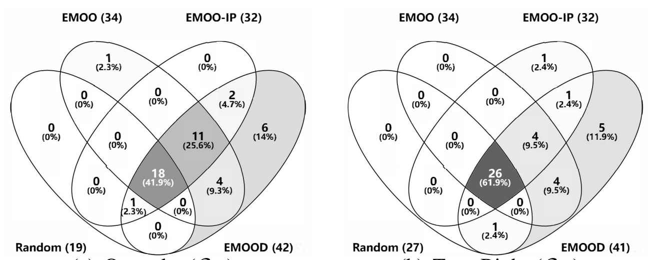
B. Results and Analysis

We next answer the two research questions and then provide qualitative analysis.

*1) Evaluation on Pattern Detection (***RQ1***):* We investigate the discovered requirements violation patterns by the different approaches and how the patterns are related. Fig. 6 shows the average number of the patterns discovered during the allocated generations. The detailed comparisons of the patterns discovered during the search process are shown in Fig. 7.

As shown in Fig. 6, we can find `EMOOD`'s average number of discovered patterns to be 30.7 and 28.8 in $\mathcal{S}_O$ and $\mathcal{S}_T$, respectively, outperforming the three baseline approaches, i.e., `Random`, `EMOO`, and `EMOO-IP`. For the first two baseline approaches `Random` and `EMOO`, there is a large increase in the number of patterns discovered within the 50th generation, while the speed to find new patterns decreases after the 100th generation. The reason is that both `EMOO-IP` and `EMOOD` could change the objective functions to search for, leading to the discovery of new patterns during the search process. Compared with the third baseline approach `EMOO-IP`, `EMOOD` adjusts the sequence of the targeting pattern to search for by also considering the likelihood to occur, thus excluding patterns that cannot occur and saving the search time. Hence, in $\mathcal{S}_O$ and in $\mathcal{S}_T$, `EMOOD` could find, respectively, 4.7 and 3.2 patterns more than `EMOO-IP`.

We also observe that `Random` performs better in $\mathcal{S}_T$ than in $\mathcal{S}_O$, as $\mathcal{S}_T$ has a smaller search space. Thus, the gaps in the



(a) Overtake ($\mathcal{S}_{\mathcal{O}}$). (b) Turn Right ($\mathcal{S}_{\mathcal{T}}$).

Fig. 7: RQ1–Comparison of discovered patterns

TABLE IV: RQ1-Time cost of the testing process

Time Cost (hours)	**Overtake ($\mathcal{S}_{\mathcal{O}}$)**				**Turn Right ($\mathcal{S}_{\mathcal{T}}$)**			
	`Rand.`	`EMOO`	`EMOO-IP`	`EMOOD`	`Rand.`	`EMOO`	`EMOO-IP`	`EMOOD`
Simulation	16.85	14.64	14.25	14.66	13.33	10.11	10.83	10.77
Computation	-	0.14	0.16	0.19	-	0.14	0.15	0.18
Total	16.85	14.78	14.41	14.85	13.33	10.25	10.98	10.95

* In the simulator, ADS_{PP} is terminated once collisions are detected.

average number of the discovered patterns between the four approaches are smaller in $\mathcal{S}_{\mathcal{T}}$ than $\mathcal{S}_{\mathcal{O}}$.

We next perform a statistical test on the final results of the approaches across the ten runs. Following a guideline [37], we use the Wilcoxon signed rank test [38] and the Vargha-Delaney's $\hat{A}_{12}$ effect size [39]. Table III reports the results of the statistical test obtained when comparing the number of the patterns discovered by `Random`, `EMOO`, `EMOO-IP`, and `EMOOD` for $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{T}}$. As shown in the table, the p-values of the results between `EMOOD` and the baseline approaches are all lower than 0.05, and the $\hat{A}_{12}$ statistics show a large effect size (close or equal to 1). Hence, the number of the patterns discovered by `EMOOD` is significantly higher than those discovered by the baseline approaches; moreover, we also notice that each technique of the approach (i.e., IP, DP, and EMOO) significantly improves over the approach without that technique.

Moreover, we want to know whether `EMOOD`'s higher effectiveness is due to its capability to discover more patterns (which do not necessarily subsume patterns discovered by the baseline approaches) or its capability to subsume all the patterns discovered by the baseline approaches plus some other patterns. We compare all the discovered patterns in the ten runs for the four approaches, as shown in Fig. 7. We find that, across ten runs, `EMOOD` could aggregately discover 42 and 41 patterns in $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{T}}$, respectively. In addition, `EMOOD` could discover 6 and 5 extra patterns (not discovered by any baseline approach), accounting for 14% and 11.9% of the total number of discovered patterns in $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{T}}$, respectively. Thus, `EMOOD`'s higher effectiveness lies in its capability to discover extra patterns compared with the baseline approaches.

Finally, to check whether the effectiveness of `EMOOD` comes at the cost of computation time, we compare the whole execution time of `EMOOD` and the baseline approaches. Considering that the simulation time of each test scenario could be large, we parallelize the simulation of the generated test scenarios using 30 threads. Table IV reports, for each approach, the time cost of the test simulations, the computation time (excluding the simulation time during fitness evaluation), and the total time cost. ADS_{PP} will be terminated in the simulation after violating some requirements, especially for $Safety(R_{2.1})$, if the ego vehicle's behaviors are abnormal. We observe that the time costs of `EMOO`, `EMOO-IP`, and `EMOOD` are less than `Random`, because these three approaches could generate more scenarios with violations of $R_{2.1}$, and these scenarios are terminated earlier. Moreover, we observe that the most expensive part of the approaches is due to the simulations.

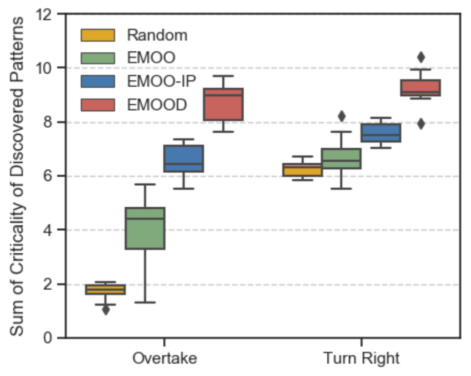


Fig. 8: RQ2–The sum of criticality of discovered patterns

Answer to RQ1: Each phase (EMOO, IP, and DP) of `EMOOD` provides a significant contribution in discovering requirements violation patterns. Indeed, the approaches of `Random`, `EMOO`, `EMOO-IP`, and `EMOOD` are in the order of increasing effectiveness, indicating that the EMOO, IP, and DP techniques all provide a relevant contribution to `EMOOD`'s overall effectiveness. Additionally, `EMOOD`'s effectiveness does not come at the cost of time.

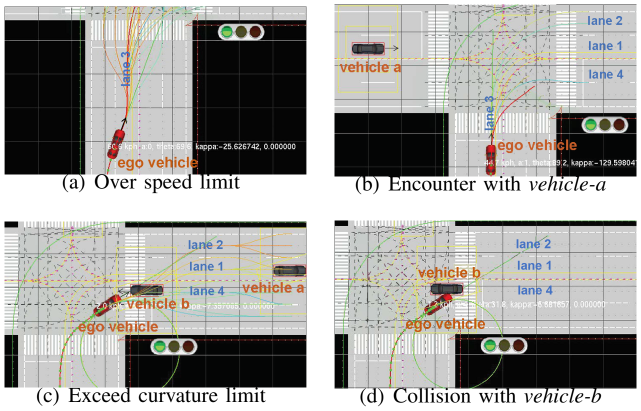
*2) Evaluation on Criticality (***RQ2***):* We analyze the criticality of requirements violation patterns discovered by each approach; to this aim, we propose a metric for assessing the criticality. As explained in Section IV-B, the initial ranking $\mathcal{V}_c$ ranks the patterns based on their criticality. Therefore, we quantify the criticality of each pattern by its rank in $\mathcal{V}_c$, i.e., $\mathcal{K}(V_i) = 1 - \frac{rank(\mathcal{V}_c(V_i))}{|\mathcal{V}_c|}$. The sum of the criticality of discovered patterns can be calculated as $\sum_{V_i \in \mathcal{V}_f} \mathcal{K}(V_i)$. Fig. 8 shows the results for the sum of the criticality of discovered patterns. We find that `EMOOD` achieves the highest sum for the criticality of discovered patterns, being 8.75 and 9.23 on average for $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{T}}$, respectively.

We also compare the results in Fig. 8, for each pair of approaches for $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{T}}$, using the same statistical tests used in RQ1. We report the results online [26] due to space limit. Using significance level α=0.01, we observe that `EMOOD` is always significantly better than the baseline approaches; `EMOO-IP` is always better than `Random`, and better than `EMOO` in $\mathcal{S}_{\mathcal{O}}$; `EMOO` is better than `Random` in $\mathcal{S}_{\mathcal{O}}$; in the other cases, there is no significant difference.

Table V provides a detailed analysis by showing the aver-

TABLE V: RQ2–Number of discovered patterns by rankings in $\mathcal{V}_c$

Pattern	**Overtake ($\mathcal{S}_{\mathcal{O}}$)**				**Turn Right ($\mathcal{S}_{\mathcal{T}}$)**			
Rankings	`Rand.`	`EMOO`	`EMOO-IP`	`EMOOD`	`Rand.`	`EMOO`	`EMOO-IP`	`EMOOD`
1-32	0.1	0.1	1.0	1.9	2.0	2.2	2.1	3.1
33-64	0.4	1.4	2.1	3.2	5.1	4.8	5.6	6.3
65-96	2.7	5.3	9.2	11.3	5.0	5.3	6.6	7.5
97-128	8.6	12.0	13.7	14.3	9.1	10.5	11.3	11.9
Total	11.8	18.8	26.0	30.7	21.2	22.8	25.6	28.8



(a) Over speed limit (b) Encounter with *vehicle-a* (c) Exceed curvature limit (d) Collision with *vehicle-b*

Fig. 9: A scenario with 4 requirements violations in $\mathcal{S}_{\mathcal{T}}$

age number of the patterns discovered by each approach in each scenario by ranks in $\mathcal{V}_c$. Compared with the baseline approaches, `EMOOD` identifies more patterns with higher ranks (i.e., ranks from 1 to 32), while the gap decreases for the patterns with lower ranks (i.e., ranks from 97 to 128). In more complex scenarios such as $\mathcal{S}_{\mathcal{O}}$, the gaps between the number of discovered patterns are more obvious between different approaches because complex scenarios might have more challenging states that are difficult to reach. We also find that the results of violation criticality are consistent with the results of pattern detection in *RQ1*. In simpler scenarios such as $\mathcal{S}_{\mathcal{T}}$, the difference between the baseline approaches and `EMOOD` is lower, showing that `EMOOD` is particularly useful in finding critical patterns in large search spaces.

Answer to RQ2: Random testing is limited in finding critical requirements violation patterns in complex scenarios, while as the combination of EMOO, IP, and DP, `EMOOD` is more effective than the baseline approaches.

3) Requirements Violation Pattern Analysis: We next analyze a concrete test scenario to better understand what the generated critical test scenarios look like and what are the behaviors and requirements evaluation results of the ego vehicle with ADS_{PP} in this scenario.

We next consider some results found by `EMOOD` in $\mathcal{S}_{\mathcal{T}}$. As shown in Fig. 7(b), `EMOOD` discovers 5 patterns not discovered by the baseline approaches; Fig. 9 reports the scenario exposing the most critical one among these patterns. In this concrete scenario, the behavior of the ego vehicle violates the majority of the requirements, i.e., $V = [1, 1, 1, 0, 0, 0, 1]$, indicating the violation of $\{R_{1.1}, R_{2.1}, R_{3.1}, R_{4.2}\}$. This pattern also ranks first in all of the discovered patterns in $\mathcal{S}_{\mathcal{T}}$. Fig. 9(a) shows the start of the scenario in which the ego vehicle is accelerating and moving into *lane-3* to turn right at the intersection quickly. In this process, the speed of the ego vehicle reaches $16.82m/s$, exceeding the speed limit ($16.67m/s$) a little, as shown in Fig. 9(a). Reacting to the violation of the speed limit, the *ego vehicle* reduces the speed when it reaches the intersection and encounters *vehicle-a* crossing from left to right in the *lane-1* at the moment shown in Fig. 9(b). To avoid the potential collision with *vehicle-a*, the *ego vehicle* decelerates a lot and turns its direction to right. However, this dangerous action causes its curvature rate to reach 0.4, which exceeds the limit of $K_{limit} = 0.2$. Fig. 9(c) shows the moment in which the *ego vehicle* reaches the maximal curvature rate at the speed of $0.56m/s$, while there is another *vehicle-b* crossing from right to left. Finally, the *ego vehicle* fails to figure out a feasible trajectory and ends with a collision with *vehicle-b* as shown in Fig. 9(d). The change rate of acceleration of this trajectory is 0.39, exceeding the threshold $r_a = 0.3$.

We next show how the patterns discovered by `EMOOD` are useful in an industrial setting, i.e., whether they provide the engineers of our industrial partner useful insights regarding the faults of ADS_{PP}. Considering the previous example, we find that, at the start of the scenario, the *ego vehicle* first accelerates to cross the intersection, and then decelerates when encountering with *vehicle-a*. However, it is too late for the *ego vehicle* to change its direction and brake. We estimate that this series of requirements violations in this scenario can be due to that the *ego vehicle* is not sensitive to exceeding the speed limit. Based on this result, we modify ADS_{PP} by penalizing the selection of paths that require the violation of the speed limit; we run the same scenario with the modified ADS_{PP}, and find that the *ego vehicle* does not violate the speed limit and, at the intersection, changes its direction into *lane-2* until *vehicle-b* has passed, and then changes into *lane-1*. The *ego vehicle* ends with no requirement violated: the maximal speed is $15.9m/s$, the maximal curvature rate is 0.13, the acceleration change rate is 0.19, and no collision occurs. Such type of information is useful for engineers of our industry partner to re-engineer ADS_{PP}.

C. Threats to Validity

The threats to external validity primarily include the degree to which the subject ADSs, their faults, and abstract test scenarios are representative of true practice. Our evaluation is conducted with only one system ADS_{PP} provided by our industry partner. Moreover, the evaluation results are based on only two common scenarios, i.e., $\mathcal{S}_O$ and $\mathcal{S}_T$, These threats could be reduced by more experiments on wider types of ADSs and more abstract scenarios in future work. The threats to internal validity are instrumentation effects that can bias our results. Faults in our prototype and the search framework jMetalPy might cause such effects. To reduce these threats, we manually inspected the intermediate results of applying our approach.

VI. Related Work

Search-Based Testing. Search-based testing (SBT) for test generation is an active area in ADS testing. The idea is to use optimization to efficiently generate critical test scenarios in which the autonomous vehicle under test fails (e.g., collision). Abdessalem et al. [6] use multi-objective search with a surrogate model [7] and learnable evolution algorithm to test an Advanced Driver Assistance System (such system supports levels 1-2 of automation [40]). For the testing of ADSs with higher-level automation and a more dedicated decision-making process, Calò et al. [10] propose an approach to generate avoidable collision scenarios in which a differently configured ADS would not lead to collisions. Gambi et al. [9] generate, through procedural content generation and search-based testing, challenging scenarios in which the vehicle under test departs the lane, while Li et al. [12] introduce AV-FUZZER, a framework based on fuzzing to find safety violations of the ADS under test. In contrast to our work, none of these preceding approaches considers the problem of multiple requirements violations, but consider the generation of critical scenarios for only one specific ADS requirement, such as safety. Furthermore, EMOOD identifies critical test scenarios covering different requirements violation patterns.

Many-objective Optimization in SBT. Many-objective search [30] is increasingly used to address the need for many objectives to represent different decision criteria involved in software engineering tasks. The usage of many-objective search presents an important challenge, since the high-dimensional Pareto fronts demand more time and memory resources to be generated. For object-oriented systems, Panichella et al. [41] design a highly scalable many-objective genetic algorithm (MOSA) for the many-objective branch coverage problem. Their later work [42] proposes DynaMOSA to dynamically focus on not-covered objectives according to branch, statement, and mutant coverage to generate tests; DynaMOSA is now the default algorithm in EvoSuite [43], an automatic test generation framework. However, the preceding existing work on many-objective optimization in search-based testing is slightly different from our approach, as we execute many rounds of EMOO, using different patterns as objectives to search for, and our prioritization aims at selecting the target patterns. Namely, based on a new insight of the relationship between requirements violation patterns, we design DP to automatically select the most likely-to-occur and critical patterns to search for at the beginning of each EMOO execution.

Coverage Criteria. Since it is infeasible to test ADSs under all possible conditions due to the large test scenario space (other vehicles, road structure, etc.), some work [44] proposes coverage criteria for ADS testing. Hauer et al. [16] define categories of different scenarios that should be covered. To test any possible behaviors of the ADS under test, Laurent et al. [13] use weight coverage to cover different configurations of a path planner, while Tian et al. [45] use neuron coverage to guide the generation of tests for neural networks in autonomous vehicles. However, neither scenario coverage nor driving behavior coverage criteria can guarantee that unsafe behaviors of the ADS are tested in terms of requirements violations, and so they cannot guarantee to expose all the possible requirements violation patterns. In contrast, EMOOD targets requirements violation patterns to identify diverse critical test scenarios for the ADS. Therefore, through the coverage of different requirements violation patterns, unsafe behaviors of the ADS are more likely to be exposed, so that ADS engineers can build a resilient ADS in extreme or stressful scenarios.

Requirements-Based Testing. Requirements-based testing for autonomous systems aims at system conformance with all identified requirements [46]. As ADSs are complex cyber-physical systems that are challenging to test and debug, requirements-based testing approaches have been proposed to check the requirements satisfaction during the development process to decrease the resources needed to design the systems [23]. To generate tests to expose requirements violations, two mainstream approaches, i.e., model checking and model testing, have been proposed for cyber-physical systems [47]. Model checking approaches [4] attempt to find failures when they exist or prove the absence of failures through formal verification. However, because formal verification considers all possible executions, it often has difficulty in scaling to complex systems with a large space of system states [27], [47]. Model testing approaches [23], [48] identify and specify requirements for the system behaviors and generate test scenarios that violate the requirements specifications. However, these approaches do not target different combinations of requirements violations, whereas EMOOD can help effectively discover the most critical and likely-to-occur requirements violation patterns.

VII. Conclusion and Future Work

In this paper, we have proposed EMOOD, an automated testing approach for ADSs, that targets requirements violation patterns. EMOOD builds on Evolutionary Many-Objective Optimization (EMOO), which is guided by Initial Prioritization (IP) and Dynamical Prioritization (DP) of requirements violation patterns to identify the violation pattern to search for during each iteration. Our evaluation results on an industrial ADS have shown that EMOOD outperforms the baseline random testing and evolutionary search, and generates test scenarios that expose more violation patterns with higher criticality.

Our current approach requires to specify, as input, an *abstract scenario* over which the search is performed. Different abstract scenarios may or may not allow to cover specific requirements violation patterns. Such manual selection of abstract scenarios requires rich domain knowledge and may limit the approach's effectiveness. In future work, we plan to investigate tool automation to identify abstract scenarios; some work [16] has been proposed to identify the *scenario types* that should be considered for ADS testing, and we plan to leverage such work to specify our initial abstract scenarios.

References

[1] U. Topcu, N. Bliss, N. Cooke, M. L. Cummings, A. Llorens, H. E. Shrobe, and L. Zuck, "Assured autonomy: Path toward living with autonomous systems we can trust," *CoRR*, vol. abs/2010.14443, 2020.

[2] Waymo, "On the Road to Fully Self-Driving: Waymo Safety Report," 2018. [Online]. Available: https://waymo.com/safety/

[3] Voyage, "Open Autonomous Safety," 2019. [Online]. Available: https://github.com/voyage/open-autonomous-safety

[4] I. Majzik, O. Semeráth, C. Hajdu, K. Marussy, Z. Szatmári, Z. Micskei, A. Vörös, A. A. Babikian, and D. Varró, "Towards system-level testing with coverage guarantees for autonomous vehicles," in *Proc. 22nd IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2019, pp. 89–94, doi: 10.1109/MODELS.2019.00-12.

[5] N. Kalra and S. M. Paddock, *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, 2016.

[6] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proc. 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 63–74, doi: 10.1145/2970276.2970311.

[7] ——, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proc. 40th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2018, pp. 1016–1026, doi: 10.1145/3180155.3180160.

[8] R. Ben Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Testing autonomous cars for feature interaction failures using many-objective search," in *Proc. 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 143–154, doi: 10.1145/3238147.3238192.

[9] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proc. 28th ACM International Symposium on Software Testing and Analysis (ISSTA)*, 2019, pp. 318–328, doi: 10.1145/3293882.3330566.

[10] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Generating avoidable collision scenarios for testing autonomous driving systems," in *Proc. 13th IEEE International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 375–386, doi: 10.1109/ICST46399.2020.00045.

[11] ——, "Simultaneously searching and solving multiple avoidable collisions for testing autonomous driving systems," in *Proc. 22nd ACM Genetic and Evolutionary Computation Conference (GECCO)*, 2020, pp. 1055–1063, doi: 10.1145/3377930.3389827.

[12] G. Li, Y. Li, S. Jha, T. Tsai, M. B. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. K. Iyer, "AV-FUZZER: finding safety violations in autonomous driving systems," in *Proc. 31st IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 25–36, doi: 10.1109/ISSRE5003.2020.00012.

[13] T. Laurent, P. Arcaini, F. Ishikawa, and A. Ventresque, "Achieving weight coverage for an autonomous driving system with search-based test generation," in *Proc. 25th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2020, pp. 93–102, doi: 10.1109/ICECCS51672.2020.00018.

[14] P. Arcaini, X.-Y. Zhang, and F. Ishikawa, "Targeting patterns of driving characteristics in testing autonomous driving systems," in *Proc. 14th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2021, pp. 295–305, doi: 10.1109/ICST49551.2021.00042.

[15] Y. Li, J. Tao, and F. Wotawa, "Ontology-based test generation for automated and autonomous driving functions," *Information and Software Technology*, vol. 117, 2020, doi: 10.1016/j.infsof.2019.106200.

[16] F. Hauer, T. Schmidt, B. Holzmüller, and A. Pretschner, "Did we test all scenarios for automated and autonomous driving systems?" in *Proc. 22nd IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 2950–2955, doi: 10.1109/ITSC.2019.8917326.

[17] C. Zhang, Y. Liu, D. Zhao, and Y. Su, "Roadview: A traffic scene simulator for autonomous vehicle simulation testing," in *Proc. 17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 1160–1165, doi: 10.1109/ITSC.2014.6957844.

[18] D. Zhao and H. Peng, "From the lab to the street: Solving the challenge of accelerating automated vehicle testing," *CoRR*, vol. abs/1707.04792, 2017.

[19] A. Belbachir, J.-C. Smal, J.-M. Blosseville, and D. Gruyer, "Simulation-driven validation of advanced driving-assistance systems," *Procedia-Social and Behavioral Sciences*, vol. 48, pp. 1205–1214, 2012.

[20] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR*, vol. abs/1708.06374, 2017.

[21] ISO, "Road vehicles – Functional safety," 2011. [Online]. Available: https://www.iso.org/standard/68383.html

[22] K. Czarnecki, "Automated driving system (ads) high–level quality requirements analysis– driving behavior safety," *Waterloo Intelligent Systems Engineering Lab (WISE) Report, University of Waterloo*, 2018.

[23] C. E. Tuncali, G. Fainekos, D. Prokhorov, H. Ito, and J. Kapinski, "Requirements-driven test generation for autonomous vehicles with machine learning components," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 265–280, 2019, doi: 10.1109/TIV.2019.2955903.

[24] J. Morse, D. Araiza-Illan, K. Eder, J. Lawry, and A. Richards, "A fuzzy approach to qualification in design exploration for autonomous robots and systems," in *Proc. 26th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017, pp. 1–6, doi: 10.1109/FUZZ-IEEE.2017.8015456.

[25] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Proc. 18th IEEE International Requirements Engineering Conference (RE)*, 2010, pp. 125–134, doi: 10.1109/RE.2010.25.

[26] EMOOD, "Project website for "Targeting Requirements Violations of Autonomous Driving Systems by Dynamic Evolutionary Search"," 2021. [Online]. Available: https://sites.google.com/view/emoodproj/

[27] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, "A survey of algorithms for black-box safety validation," *CoRR*, vol. abs/2005.02979, 2020.

[28] S. Chand and M. Wagner, "Evolutionary many-objective optimization: A quick-start guide," *Surveys in Operations Research and Management Science*, vol. 20, no. 2, pp. 35–42, 2015.

[29] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2013, doi: 10.1109/TEVC.2013.2281535.

[30] A. Ramirez, J. R. Romero, and S. Ventura, "A survey of many-objective optimisation in search-based software engineering," *Journal of Systems and Software*, vol. 149, pp. 382–395, 2019, doi: 10.1016/j.jss.2018.12.015.

[31] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *Proc. 31st International Conference on Computer-Aided Verification (CAV)*, 2019, pp. 432–442, doi: 10.1007/978-3-030-25540-4_25.

[32] F. Klück, M. Zimmermann, F. Wotawa, and M. Nica, "Performance comparison of two search-based testing strategies for ADAS system validation," in *Proc. 31st IFIP International Conference on Testing Software and Systems (ICTSS)*, 2019, pp. 140–156, doi: 10.1007/978-3-030-31280-0_9.

[33] T. G. Dietterich, "Ensemble methods in machine learning," in *Proc. 1st International Workshop on Multiple Classifier Systems Multiple (MCS)*, 2000, pp. 1–15, doi: 10.1007/3-540-45014-9_1.

[34] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys*, vol. 45, no. 1, pp. 11:1–11:61, 2012, doi: 10.1145/2379776.2379787.

[35] A. Benitez-Hidalgo, A. J. Nebro, J. Garcia-Nieto, I. Oregi, and J. Del Ser, "jMetalPy: A Python framework for multi-objective optimization with metaheuristics," *Swarm and Evolutionary Computation*, vol. 51, 2019, doi: 10.1016/j.swevo.2019.100598.

[36] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[37] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd IEEE/ACM International Conference on Software Engineering (ICSE)*, 2011, pp. 1–10, doi: 10.1145/1985793.1985795.

[38] J. A. Capon, *Elementary Statistics for the Social Sciences: Study Guide*. Wadsworth Publishing Company Belmont, 1991.

[39] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000, doi: 10.3102/10769986025002101.

[40] SAE, “Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles,” 2018. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/

[41] A. Panichella, F. M. Kifetew, and P. Tonella, “Reformulating branch coverage as a many-objective optimization problem,” in *Proc. 8th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–10, doi: 10.1109/ICST.2015.7102604.

[42] ——, “Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets,” *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 122–158, 2017, doi: 10.1109/TSE.2017.2663435.

[43] G. Fraser and A. Arcuri, “EvoSuite: automatic test suite generation for object-oriented software,” in *Proc. 19th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 416–419, doi: 10.1145/2025113.2025179.

[44] Z. Tahir and R. Alexander, “Coverage based testing for V&V and safety assurance of self-driving autonomous vehicle: A systematic literature review,” in *Proc. 2nd IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2020, pp. 23–30, doi: 10.1109/AITEST49225.2020.00011.

[45] Y. Tian, K. Pei, S. Jana, and B. Ray, “DeepTest: Automated testing of deep-neural-network-driven autonomous cars,” in *Proc. 40th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2018, pp. 303–314, doi: 10.1145/3180155.3180220.

[46] R. Alexander, H. Hawkins, and A. Rae, *Situation coverage – a coverage criterion for testing autonomous robots*. Department of Computer Science, University of York, 2015, vol. Report number YCS-2015-496.

[47] S. Nejati, K. Gaaloul, C. Menghi, L. C. Briand, S. Foster, and D. Wolfe, “Evaluating model testing and model checking for finding requirements violations in Simulink models,” in *Proc. 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019, pp. 1015–1025, doi: 10.1145/3338906.3340444.

[48] C. Gladisch, T. Heinz, C. Heinzemann, J. Oehlerking, A. von Vietinghoff, and T. Pfitzer, “Experience paper: search-based testing in automated driving control applications,” in *Proc. 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 26–37, doi: 10.1109/ASE.2019.00013.