LeGEND: A Top-Down Approach to Scenario Generation of Autonomous Driving Systems Assisted by Large Language Models

Anonymous Author(s)

ABSTRACT

Autonomous driving systems (ADS) are safety-critical and require comprehensive testing before their deployment on public roads. While existing testing approaches primarily focus on the criticality of scenarios, they often overlook the diversity of the generated scenarios that is also important to reflect system defects in different aspects. To bridge the gap, we propose LeGEND, that features a top-down fashion of scenario generation: it starts with abstract functional scenarios, and then step downwards to logical and concrete scenarios, such that scenario diversity can be controlled at the functional level. However, unlike logical scenarios that can be formally described, functional scenarios are often documented in natural languages (e.g., accident reports) thus cannot be precisely parsed and processed by computers; to tackle that issue, LeGEND leverages the recent advances of large language models to transform functional scenarios in natural languages to formal logical scenarios. To mitigate the distraction of useless information in functional scenarios, we devise a two-stage transformation that features the use of an intermediate language; consequently, we adopt two LLMs in LeGEND, one for extracting information from functional scenarios, the other for converting the extracted information to formal logical scenarios. We experimentally evaluate LeGEND on Apollo, an industry-grade ADS from Baidu. Evaluation results show that LeGEND can effectively identify critical scenarios, and compared to baseline approaches, LeGEND exhibits evident superiority in diversity of generated scenarios. Moreover, we also demonstrate the advantages of our two-phase transformation framework, and the accuracy of the adopted LLMs.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; **Search-based software engineering**.

KEYWORDS

autonomous driving systems, scenario generation, large language model, search-based testing

1 INTRODUCTION

Autonomous driving systems (ADS) have been recognized as a revolutionary advancement in automotive industry, offering a new solution to reducing mistakes by human drivers and mitigating traffic congestion. These systems bring unprecedented driving experience and significant efficiency, by employing intelligent sensors to perceive surrounding environments and sophisticated algorithms to make control decisions. However, despite these advantages, malfunctions of ADS are extremely dangerous, potentially leading to catastrophic consequences that pose fatal threats to human lives [10]. To ensure the safety of ADS, systematic testing is an indispensable stage before their deployment on public roads.

Given that real-world testing of ADS is prohibitively expensive, scenario-based testing in simulation environments has been widely adopted, thanks to its superior flexibility [38]. The goal of this approach is to generate critical scenarios, e.g., in which ADS collide with other vehicles, so as to expose potential system defects and assist engineers to seek for effective remedies. The state-of-the-practice [9, 16, 21, 39, 40, 46, 47] often starts with a *logical scenario* that specifies environments (e.g., road structure, weather) and traffic participants, but leaves a state space open (identified by a number of variables, e.g., initial states of vehicles), such that critical *concrete scenarios* can be detected there by optimization-based search.

While these approaches are effective to detect critical scenarios, they also suffer from a severe issue about the *diversity* of detected scenarios, i.e., the detected scenarios may exhibit similar behaviors of ADS that reflect similar system defects. Even though some approaches [9, 40, 47] mitigate this issue by taking diversity of scenarios as a search objective, the effect is still limited, since the detected scenarios remain constrained by logical scenarios.

Motivations and Challenges Compared to logical level, *functional* level offers even more abstraction of scenarios that captures only their conceptual features. For instance, in Fig. 1, while the logical scenario has strict restrictions on the space in which concrete scenarios can vary, functional scenarios just give a conceptual description about the featured events of scenarios. To that end, starting scenario generation with functional level is a plausible way to facilitate a great diversity of generated scenarios.

However, a challenging issue also arises in that direction: unlike logical scenarios that can be formally represented by *domain-specific languages (DSL)*, functional scenarios are described conceptually at a more abstract level; consequently, they are often documented informally in natural languages. Given that the final objectives (i.e., concrete scenarios) of scenario-based testing are still formally characterized, we need to devise a systematic approach that can accommodate natural language inputs of functional scenarios.

Contributions We propose an approach LeGEND (<u>L</u>arge Language Model <u>E</u>nabled <u>G</u>eneration of Sc<u>EN</u>ario for Testing of the Autonomous <u>D</u>riving Systems) that exploits the recent advances of *large language models (LLMs)* to handle natural language inputs, and can generate scenarios that are not only critical but also diverse. Specifically, LeGEND adopts a top-down scheme that generates scenarios across three different abstraction layers of scenarios: it starts with abstract functional scenarios documented in natural

ASE'24, Sacramento, California, USA

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/nnnnnnn.nnnnnnn

languages and transforms it to logical scenarios that can be represented in domain specific languages; then, it searches in the open space of logical scenarios for critical concrete scenarios. Thanks to the top-down scheme, LeGEND can effectively control, and thereby achieve a great diversity of generated scenarios.

To avoid unrealistic scenarios at the functional level, LeGEND selects functional scenarios from real-world accident reports [29]. However, this raises another concern about the precision of LLMs, because there could be much distracting information in those accident reports. To that end, instead of using LLM directly, we devise a two-phase transformation with the assistance of an intermediate representation, called *interactive pattern sequence (IPS)*, that can record the featured events and their logical relations in reports. Consequently, LeGEND involves two LLMs, i.e., LLM_1 that extracts featured events in accident reports into IPS, and LLM_2 that converts IPS to logical scenarios in formal DSL representations.

We evaluate LeGEND on Apollo [5], which is an industry-grade ADS from Baidu. Experimental results show that, compared with two baseline approaches, LeGEND can generate a more diverse set of critical scenarios. We also study the effectiveness of the intermediate representation (i.e., IPS), and the results show that it is indeed useful to improve the precision of transformation from natural language to formal scenarios. To understand the performance of the two LLMs adopted in LeGEND, we also perform a user study that asks users to rate the performances of the LLMs in their respective tasks. Our study receives a positive feedback about the performance of LLMs, which signifies their strengths in accomplishing these tasks and great potentials in ADS testing.

In summary, this paper makes the following contributions:

- We propose LeGEND, a top-down scenario generation approach that can achieve both criticality and diversity of scenarios;
- We devise a two-stage transformation, by using an intermediate language, from accident reports to logical scenarios; so, LeGEND involves two LLMs, each in charge of one different stage;
- We implement LeGEND and demonstrate its effectiveness on Apollo, and we detect 11 types of critical concrete scenarios that reflect different aspects of system defects. All experiment results and the source code are available in [4].

2 BACKGROUND

We first introduce scenarios and their abstraction hierarchy, and then we review the scenario-based testing approach of ADS.

2.1 Abstraction Hierarchy of Scenarios

Scenario is a commonly-used term in ADS testing. According to Ulbrich et al. [41], it refers to a collection of actors (including ADS and other traffic participants, e.g., *non-player character (NPC)* vehicles and pedestrians) attached with goals, actions and events, and their function environment (e.g., road structure, weather.).

As defined by Menzel et al. [26], based on their levels of abstraction, scenarios can be categorized into *functional scenarios*, *logical scenarios*, and *concrete scenarios*, as explained as follows:

- *Functional scenarios* involve semantic descriptions about the participant entities and their interrelations;
- *Logical scenarios* provide parameters and their respective ranges to characterize the state space of a functional scenario;

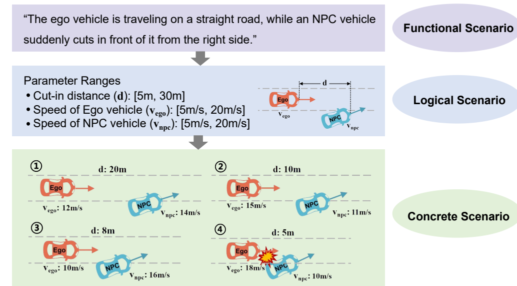


Figure 1: Scenarios in different abstraction levels

- *Concrete scenarios* are identified by assigning concrete values for each of the parameters in a logical scenario.

These three levels of abstraction offer a structured and systematic way that facilitates the safety assurance practice throughout the development of ADS [25]. More details can be found in §2.2.

We present an example in Fig. 1 to illustrate the scenarios in different abstraction levels. As shown in Fig. 1, a functional scenario is a conceptual description about a scenario: *"the ego vehicle is traveling on a straight road, while an NPC vehicle suddenly cuts in front of it from the right side"*. This functional scenario features an action *"cut-in from right side"* from the NPC vehicle; however, it can still vary w.r.t. many factors, e.g., *the cut-in distance* and *the respective speed of the ego and NPC vehicles when cut-in happens*.

By selecting parameters in a functional scenario and fixing a range for each of them, we can obtain a logical scenario, which constrains the variation space of scenarios (i.e., they can only vary w.r.t. the selected parameters and within the specified ranges). In Fig. 1, the logical scenario is obtained by selecting three parameters in the functional scenario and specifying a range for each of them.

Then, by assigning different values for the parameters in a logical scenario, we can identify different concrete scenarios. A concrete scenario can be executed in a software simulator. Essentially, the aim of ADS testing is to find such concrete scenarios that can expose dangerous behaviors (e.g., collision with other vehicles) of ADS.

2.2 Scenario-Based Testing of ADS

Scenario-based testing has been a widely-adopted approach for safety assurance of ADS [38]. To expose dangerous behaviors of ADS, it runs a variety of scenarios in software simulators and assesses their safety performances. In a plethora of literature [9, 16, 21, 39, 40, 46, 47], scenario-based testing starts with a logical scenario that has a set of searchable parameters P concerning with the states of traffic participants, e.g., *initial states of the ego vehicle* and *actions of NPC vehicles*. Then, the search problem for critical scenarios can be cast to the following (multi-objective) optimization problem:

$$P^* = \operatorname*{argmin}_P \begin{cases} \mathcal{F}_1(P) \\ \vdots \\ \mathcal{F}_N(P) \end{cases} \tag{1}$$

where $\mathcal{F}_i$ ($i \in \{1, \ldots, N\}$) is an objective function (a.k.a. *fitness* function) that formalizes a specific search goal, including not only metrics for assessing the criticality of the scenarios, such as the minimal distance between vehicles and minimal *time-to-collision* [40], but also specific user preferences about scenarios, e.g., Calo et al. [7] prefers the collisions that are avoidable.

The optimization problem aims to search for an optimal parameter P^* that minimizes the values of the objective functions. However, it may not be feasible to optimize all the objective functions at the same time, because these different search objectives may conflict with each other, in the sense that the decrease of one function may lead to the increase of another function. To that end, solvers for multi-objective optimization often aim to search for a *Pareto-front* that consists of multiple solutions, each of which is optimal in terms of at least one objective function. This have been many established solvers, such as the widely-adopted one NSGA-II [11].

As explained in §1, diversity of generated scenarios is an important property to pursue in ADS testing, in order to expose different types of system defects and avoid redundant exploration of search space. However, the existing approaches in Eq. 1 often fail to generate scenarios with high diversity, because they do not explicitly take it into account in the design of testing approaches. Some techniques [9, 40, 47] take a measure to mitigate this issue, by setting one of the objective functions to be a diversity metric; while it helps to improve diversity, the effect is rather limited because the diversity of scenarios is still bounded by the high-level logical scenarios.

3 APPROACH OVERVIEW

In this paper, we propose an approach LeGEND to scenario generation of ADS, which aims at not only criticality, but also diversity of generated scenarios. LeGEND features a top-down style of scenario generation: unlike existing approaches that start scenario generation with the logical level, LeGEND starts with the functional level that is more abstract and thereby covers a much broader range of scenarios naturally, and then steps downwards to the logical level to search for critical concrete scenarios. In this way, LeGEND can explore the scenario space at an abstract level and thereby hold a high-level control on the diversity of scenarios.

Unlike logical scenarios that can be expressed formally by DSL, functional scenarios are often documented informally by natural languages. As a consequence, it raises a challenging issue about the transformation from natural language descriptions to logical scenarios in formal DSL, where the latter are necessary for computers to automatically parse and search for concrete scenarios. To tackle this issue, we leverage the recent advances of *large language models (LLMs)*, which have demonstrated unprecedented performances in cognition, logical reasoning and natural language comprehension.

Overview of LeGEND The workflow of LeGEND is depicted in Fig. 2, which consists of three phases. In this workflow, the first two phases are in charge of the transformation from natural language that documents functional scenarios to logical scenarios in formal DSL, and the last phase employs a search-based technique similar to existing studies (see §2.2). In the following, we elaborate on the design of the first two phases, as they are the main contributions.

To avoid unrealistic scenarios that may never happen in real world, LeGEND adopts functional scenarios documented in real accident reports. Specifically, we take these reports from [29], which are official documents used to record the real accidents that happened in United States during the years 2005-2007. The contents of these reports describe the factual information related to accidents such as car crashes, with detailed time, location, traffic participants and the series of events happening before the crash.

In transformation of these reports to logical scenarios, a challenging issue arises about the precision of the transformation, since the reports are often lengthy and may contain considerable irrelevant information. To that end, instead of using one LLM directly, we adopt an intermediate representation, called *interactive pattern sequence (IPS)*, to record the featured events in the report and the logical relation over the events. In our empirical evaluation (in §6.2), we show that IPS indeed play an important role in improving the precision of transformation, by a comparison with a straightforward transformation without using IPS. As a result, we employ two LLMs, respectively in the first two phases of LeGEND, as follows:

- In Phase 1, LLM_1 extracts the useful information in an accident report and records them into an IPS;
- In Phase 2, taking an IPS as input, LLM_2 translates it into a logical scenario represented in DSL.

Example We use an example, shown in Fig. 3, to illustrate how LeGEND transforms a real accident report to a logical scenario. The description of the accident is in the top plot of Fig. 3b, with an illustration in Fig. 3a. The accident involves three vehicles, denoted as $V1$, $V2$ and $V3$, driving in three different lanes at the initial stage. Then, $V1$ wants to change to lane 1 by overtaking $V2$, but due to their close distance, $V1$ collides with $V2$; due to the impact the collision, $V2$ swerves left and hits $V3$ in lane 3 subsequently.

In Phase 1, LLM_1 takes such an accident report and transforms it to an intermediate representation, shown as the middle plot of Fig. 3b. In this representation, we can see that it involves all necessary ingredients that characterize the accident, including the initial status of the vehicles, the interactions between vehicles that lead to collisions, and their chronological relations embodied by the order of statements. This representation provides a concise format for recording the key information in accident reports, facilitating the subsequent transformation to formal representations.

Having such an intermediate representation, in Phase 2, LLM_2 converts it to a logical scenario expressed by a formal DSL. Since the adopted DSL [39] also represents actions of vehicles in a sequential manner, LLM_2 can reliably accomplish the transformation.

By these two phases, we obtain a logical scenario that faithfully reflects the ingredients in the functional scenario from real accident report, and so we can proceed with classic search techniques for detecting critical scenarios. In this loop, the diversity of scenarios can be controlled at the functional level, so we can devise our strategy to explore conceptually different scenarios on demand.

4 DETAILS OF THE PROPOSED APPROACH

In this section, we elaborate on the technical details of the three phases of LeGEND, respectively in §4.1, §4.2 and §4.3.

4.1 Extraction from Accident Reports

As introduced in §3, Phase 1 employs LLM_1 that takes an accident report as input and produces an interactive pattern sequence and other information such as road structures. We take a further look into the formats of the input and output of LLM_1.

To avoid unrealistic scenarios that may never happen in real world, we extract functional scenarios from public accident reports

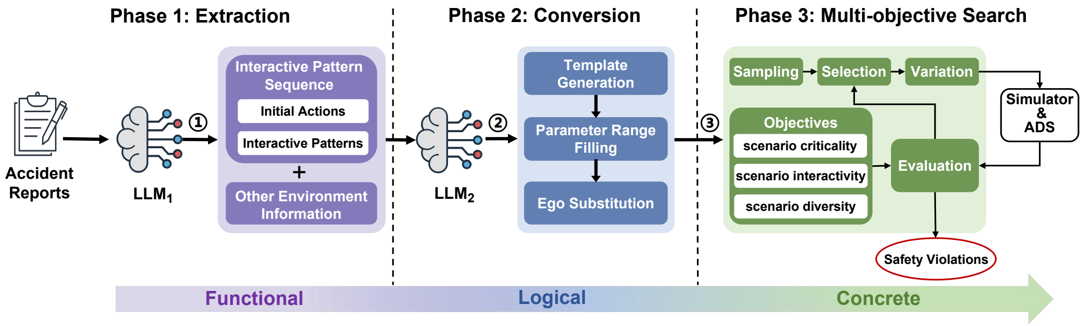
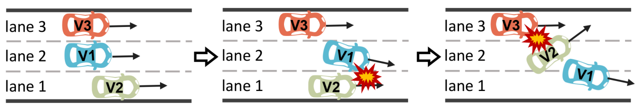
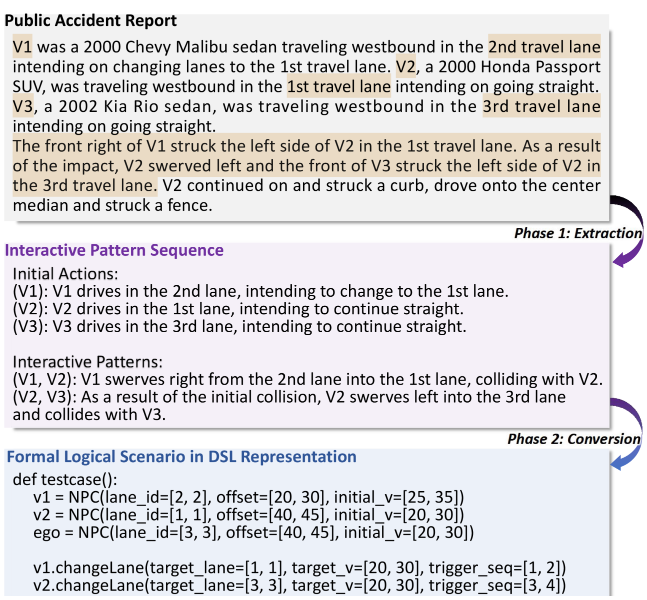


Figure 2: The workflow of LeGEND



(a) An illustration of a real accident [28]



(b) Transformation from the accident report to logical scenario

Figure 3: An example of transforming an accident report to a logical scenario in DSL

were collected from the *National Highway Traffic Safety Administration (NHTSA)* database [29]. These reports document real accidents that happened in United States. The reports mainly consist of detailed descriptions about the environments, traffic participants, and a series of events (i.e., actions of different participants) preceding the accident. The accidents often take place on two types of road structures, i.e., straight and curved roads, and include multiple types of crashed, such as *rear-end collisions* between multiple vehicles.

LLM_1 mainly outputs an intermediate representation called *interactive pattern sequences*, which involve a tuple $\langle D_\text{I}, D_\text{P}\rangle$, where D_I denotes the initial actions of vehicles, and D_P describes the interactive patterns between vehicles before accident.

- D_I is a sequence initial actions, with each action formatted as follows, where V_i is a vehicle, followed by a natural language description about the states and the intended action of V_i.

$$\langle V_i \rangle : \{\text{initial states and actions of } V_i\}$$

- D_P is a sequence of interactive patterns, with each pattern formatted as follows, where V_i and V_j are two vehicles, followed by a description about the actions of V_i and V_j w.r.t. their interaction:

$$\langle V_i, V_j \rangle : \{\text{actions of } V_i \text{ and } V_j \text{ related to interaction}\}$$

Note that, the verbs used to describe actions are required to be selected from the set *{brake, decelerate, accelerate, swerve left/right}*. Typically, an interactive pattern comes into being when one vehicle performs an active action and another vehicle follows a responsive action.

Prompt engineering Interactive pattern sequences are generated via prompt engineering applied to the LLM_1. Here, we provide an example to illustrate how LLM_1 is instructed for extracting an interactive pattern sequence from a given accident report. Specifically, we give the following extraction instruction to LLM_1:

> Prompt for LLM_1
>
> ***Task:*** *Please extract the road structure, and the interactive pattern sequence (IPS) from the given accident report.*
> ***IPS Format:***
> *V_i: initial action of V_i.*
> *V_j: initial action of V_j.*
> *(V_i, V_j): interactive actions between V_i and V_j.*
> ***Attentions:*** A1, A2, ...

The prompt applied to LLM_1 is structured into three parts: 1) the first part specifies the task assigned to LLM_1, namely, the extraction of interactive pattern sequence (IPS) and other information from the given accident report; 2) the second part provides a detailed template for the required IPS format, including the initial actions of the involved vehicles, followed by their interactive actions; 3) the third part outlines specific attentions that LLM_1 must adhere to, during the extraction, such as the requirement regarding the verbs they must select from the predefined set, as aforementioned.

Moreover, the extracted interactive pattern sequences are subject to a legality check to ensure their correct formatting. If the format is incorrect, LLM_1 is instructed to repeat the process. Once they can pass the legality check, these interactive pattern sequences are used as inputs for the subsequent phase of logical scenario conversion.

4.2 Conversion to Logical Scenario

In Phase 2, the conversion from an interactive pattern sequence to a logical scenario involves three stages, i.e., template generation (§4.2.1), parameter range filling (§4.2.2), and ego substitution (§4.2.3). Among these stages, the first two stages are generation tasks and so we employ LLM_2 to accomplish; the last stage can be performed locally. The outcome of this phase is a logical scenario in formal DSL, which can be automatically parsed and processed, as an input of Phase 3 for search of concrete scenarios.

4.2.1 Template Generation. As introduced in §2 and §3, a logical scenario can be formally represented in DSL. In this paper, we adopt a DSL in [39] that features a sequential representation of test cases (i.e., scenarios in our context), thus in line with the formats of interactive pattern sequences. We briefly introduce this DSL.

In this DSL, a test case S is a sequence of initial states of vehicles and actions of NPC vehicles, as follows:

$$\mathsf{S} = \{\underbrace{\mathsf{I}_1, \ldots, \mathsf{I}_m}_{\text{initial states}}, \underbrace{\mathsf{A}_1, \ldots, \mathsf{A}_n}_{\text{actions}}\} \tag{2}$$

where I_i ($i \in \{1, \ldots, m\}$) is the initial states of the i-th vehicle, and A_j ($j \in \{1, \ldots, n\}$) is an action of an NPC vehicle. Specifically, the actions we consider include *accelerating*, *decelerating*, and *lane changing*, each attached with different types of parameters.

From the perspective of implementation, both I_i and A_j are function calls, so they allow different types of parameters. The details of valid parameters for different actions are listed in Table 1.

The task of LLM_2 in this stage is to convert an interactive pattern sequence to a *template* of such a test case, namely, while the names of the functions in S are fixed, the parameters are unspecified. The parameter ranges are later decided as presented in §4.2.2.

Prompt engineering Since the DSL we adopted is a formal language, it is necessary to ensure that LLM_2 can generate test cases with valid syntax. To that end, we employ *one-shot learning* to instruct LLM_2 to accomplish the generation task. One-shot learning is a promising way [45] to train LLMs by prompt engineering: unlike cumbersome model training (e.g., *fine-tuning* in LLM), one-shot learning can teach LLMs to learn desired output formats by feeding examples in prompts. In our case, since the format of interactive pattern sequences is close to the formalism of our adopted DSL, lightweight methods like one-shot learning could suffice.

Specifically, we give the following instruction to LLM_2:

> Prompt for LLM_2
>
> ***Task:*** *Please generate the test case template corresponding to the given functional scenario.*
> ***Test Case Model & Example:*** *S*
> ***Attentions:*** *A1, A2, ...*

Similar to the one in §4.1, it also consists of a task instruction, a list of attentions; the differences mainly involves the example of the formalism of DSL, in the part of ***Test Case Model & Example***. This example provides the description of the test case model as well as a test case template in the form of Eq. 2, which can be learned by LLM_2.

Table 1: Test Case Model

Type	Name	Parameter
Constructor	NPC	Lane id, lane offset, initial speed
Method	Accelerate	Target speed, trigger sequence
Method	Decelerate	Target speed, trigger sequence
Method	Lane changing	Target lane, target speed, trigger sequence

4.2.2 Parameter Range Filling. In this stage, we fill the parameter ranges in the generated template. This can also be done by LLMs with information about road specifications and default parameters. However, by our observation (outlined in §6.3), the parameter ranges filled by LLMs may not be entirely rational. For instance, consider two vehicles initially positioned in the same lane. The LLM might generate parameter ranges like $[0, 5]$ and $[5, 10]$ for their positions. Since each vehicle has its own length, these ranges could result in a collision at the beginning of simulation. As a countermeasure, LeGEND retains some of the parameter ranges provided by LLMs (in particular discrete parameters, e.g., the target lane for a lane change action), but also adopts default ranges for other parameters that represent continuous variables.

4.2.3 Ego Substitution. To apply the scenarios in ADS testing, we need to assign one vehicle involved as the ego ADS vehicle. Given that an accident typically involve multiple vehicles, the ego vehicle can be any one involved in the accident. To assess ADS in more challenging situations, we retain those vehicles that perform active actions (that finally lead to accidents) as NPCs, and substitute the one that is relatively passive in the accident as the ego vehicle. To identify this vehicle, we analyze the interactive pattern sequences by counting the frequency of active actions performed by each all involved vehicles. The vehicle with the lowest frequency of active actions is then selected as the ego vehicle. This selected NPC vehicle is then designated as the ego vehicle, and its actions are removed from the test case template to hand over control to ADS algorithms.

4.3 Search-Based Concrete Scenario Generation

Having logical scenarios generated from §4.2, we leverage a multi-objective genetic algorithm, as presented in Alg. 1, to search for critical concrete scenarios. Overall, the algorithm consists of three main steps: *initial sampling* (Line 5-Line 11), *pareto-optimal selection* (Line 20), and *genetic variation* (Line 12-Line 19). Next, we introduce these steps in detail.

4.3.1 Initial Sampling. First, LeGEND constructs an initial population set consisting of random samplings of concrete scenarios. By filling each statement in a logical scenario with a randomly generated parameter value sampled from the corresponding range (Line 9), it constructs a concrete scenario (Line 10). The generated concrete scenarios are then added into a temporary set (Line 11).

4.3.2 Pareto-optimal Selection. To guide the search, we adopt three objective functions, including *scenario criticality*, *scenario interactivity*, and *scenario diversity*, adapted from literature [39, 40]. Due to these different objective functions, we need to search for and

Algorithm 1 Multi-objective search algorithm

Input: Population size p_{max}, logical scenario LS and size s_{max}, maximum number of generations g_{max}
Output: Critical test case set $\mathcal{V}$

1: $\mathcal{V} \leftarrow \emptyset$ ▷ the set of critical scenarios
2: $\mathcal{P} \leftarrow \emptyset$ ▷ the population during search
3: **for** $g \in \{1, \ldots, g_{max}\}$ **do**
4: $\quad \mathcal{P}' \leftarrow \emptyset$
5: $\quad$ **if** $g = 1$ **then** ▷ the 1st generation
6: $\quad\quad$ **while** $|\mathcal{P}'| < p_{max}$ **do**
7: $\quad\quad\quad \mathrm{CS} \leftarrow \emptyset$
8: $\quad\quad\quad$ **for** $i \in \{1, \ldots, s_{max}\}$ **do**
9: $\quad\quad\quad\quad \mathrm{CS}(i) \leftarrow \textsc{Fill}(\mathrm{LS}(i))$
10: $\quad\quad\quad\quad \mathrm{CS} \leftarrow \mathrm{CS} \cup \{\mathrm{CS}(i)\}$
11: $\quad\quad\quad \mathcal{P}' \leftarrow \mathcal{P}' \cup \{\mathrm{CS}\}$
12: $\quad$ **else** ▷ from the *2nd* generation on
13: $\quad\quad$ **while** $|\mathcal{P}'| < p_{max}$ **do**
14: $\quad\quad\quad$ tournament selection of $\mathrm{CS}_1, \mathrm{CS}_2 \in \mathcal{P}$ ▷ selection
15: $\quad\quad\quad$ randomly select index j ▷ crossover
16: $\quad\quad\quad \mathrm{CS}'_1, \mathrm{CS}'_2 \leftarrow$ swap the jth npc in $\mathrm{CS}_1, \mathrm{CS}_2$
17: $\quad\quad\quad$ **for** $i \in \{1, \ldots, s_{max}\}$ **do** ▷ mutation
18: $\quad\quad\quad\quad$ mutate parameters of $\mathrm{CS}'_1(i)$ and $\mathrm{CS}'_2(i)$
19: $\quad\quad\quad \mathcal{P}' \leftarrow \mathcal{P}' \cup \{\mathrm{CS}'_1, \mathrm{CS}'_2\}$ ▷ add $\mathrm{CS}'_1, \mathrm{CS}'_2$ to $\mathcal{P}'$
20: $\quad \mathcal{P} \leftarrow \arg_{\mathrm{CS} \in \mathcal{P} \cup \mathcal{P}'} \begin{cases} \min f_{\mathrm{MHD}} \\ \max f_{\mathrm{ACR}} \\ \max f_{\mathrm{DIV}} \end{cases}$ ▷ Pareto-optima
21: $\quad$ **for** $\mathrm{CS} \in \mathcal{P}$ **do**
22: $\quad\quad$ **if** $f_{\mathrm{MHD}}(\mathrm{CS}) < l$ **then** ▷ critical scenario found
23: $\quad\quad\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{\mathrm{CS}\}$ ▷ return critical ones

select the Pareto-optimal solutions. Below, we elaborate on the definitions of these objective functions, under the assumption that the simulation is executed over a time interval $[0, T]$.

Scenario Criticality Given that the primary objective is to identify critical scenarios that can expose safety violations, the first objective function is the *Minimum Headway Distance* (*MHD*) [2]. This metric measures the minimum headway distance of the ego vehicle to other NPCs during the simulation. Given a set V_{N} of NPC vehicles, the f_{MHD} is calculated as follows:

$$f_{\mathrm{MHD}} := \min_{t \in [0,T]} \min_{\mathrm{v} \in V_{\mathrm{N}}} \mathsf{Dis}(\mathsf{Head}_{\mathsf{ego}}, \mathsf{Head}_{\mathsf{v}})$$

where $\mathsf{Head}_{\mathsf{ego}}$ and $\mathsf{Head}_{\mathsf{v}}$ represent the headway positions of the ego vehicle and an NPC vehicle v respectively, and Dis calculates the Euclidean distance between the two positions. A smaller f_{MHD} corresponds to a higher fitness score; when the f_{MHD} falls below a threshold value l, such as the length of a vehicle, it indicates a collision between the ego vehicle and NPC vehicles, thereby we can classify it as a critical scenario (Line 21-Line 23).

Scenario Interactivity With this objective, we aim to produce critical scenarios by enhancing the interactivity between the ego vehicle and NPC vehicles. We adopt the metric *Acceleration Change Rate* (*ACR*) [40], which quantifies the interactivity between vehicles by measuring the rate of change in the ego vehicle's acceleration in response to the behaviors of NPC vehicles. f_{ACR} is as follows:

$$f_{\mathrm{ACR}} := \frac{1}{T}\left(\sum_{t_1, t_2 \in [0,T]} \mathbb{1}\left(\left|a(t_1) - a(t_2)\right| \geq \eta\right)\right) \quad \text{s.t.} \quad \dot{a}(t_1) = 0, \quad \dot{a}(t_2) = 0$$

where a represents the acceleration of the ego vehicle, $\dot{a}$ denotes its first-order derivative, and $\mathbb{1}$ is the indicator function with η as a threshold. The indicator function returns 1 if the specified condition is met and 0 otherwise. The more frequent interactions are, the greater the value of f_{ACR} is.

Scenario Diversity This objective is to enrich the diversity of scenarios under the same logical scenarios; to that end, we adopt f_{DIV}, which measures the average Euclidean distances over detected scenarios, by treating each scenario as a point (the dimension of which is the same to the number of parameters in the logical scenario). A larger f_{DIV} value indicates a broader exploration in the parameter configuration space.

We then proceed to identify Pareto-optimal test cases based on their values across the three objectives. Specifically, all of the test cases that exhibit superior performance in at least one objective are retained within the population. These selected test cases will undergo a genetic variation phase, as detailed in §4.3.3, to generate new candidate test cases for the subsequent generations.

4.3.3 Genetic Variation. From the second generation on (Line 12), our algorithm, as illustrated in Alg. 1, constructs a candidate set of test cases based on those selected Pareto-optima in §4.3.2. This process involves two types of genetic variation operations: *crossover* and *mutation*. These operations are designed to handle the sequential test case model, as described subsequently, and ensure effective exploitation by the algorithm.

Crossover In each iteration, this operation acquires two test cases CS_1 and CS_2 through tournament selection, based on the evaluation of the objective function (Line 14). Subsequently, a vehicle index j is randomly selected, and the j-th NPC vehicle, including its initial positions and driving maneuvers, is tentatively swapped between the two concrete test cases. This process will yield two new test cases CS'_1 and CS'_2 (Line 16).

Mutation For each statement in CS'_1 and CS'_2, this process will apply a polynomial mutation [46] to each discrete and continuous variable. Note that the bounds for parameters have been fixed as presented in the logical scenarios.

5 EXPERIMENT DESIGN

Research Questions To evaluate the performance of LeGEND, we investigate the following research questions:

- **RQ1:** *Can LeGEND improve diversity of scenarios, compared to baseline approaches?* In this RQ, we investigate the diversity of critical scenarios detected by LeGEND. Moreover, We compare LeGEND with baseline approaches under different metrics.
- **RQ2:** *Can interactive pattern sequences help to improve transformation precision?* In this RQ, we study the usefulness of interactive pattern sequences for transformation precision.
- **RQ3:** *How accurate are* LLM_1 *and* LLM_2 *for the transformation tasks in LeGEND?* In this RQ, we study the accuracy of LLM_1 for extraction in Phase 1 and LLM_2 for conversion in Phase 2.

Baselines and metrics To address RQ1, we perform a comparative analysis with two baseline approaches: *Random* and *AV-Fuzzer* [21]. The *Random* approach generates NPC vehicles and their driving actions randomly. *AV-Fuzzer* initially generates scenarios randomly, and then employs a genetic algorithm to search for more critical scenarios. However, *AV-Fuzzer* has limited ability to vary the number of NPC vehicles and select different road structures during testing. Therefore, for comparison purposes, we evaluate *AV-Fuzzer* in a fixed road environment with a constant number of NPC vehicles, such as a straight road with two NPC vehicles.

We run LeGEND, *Random*, and *AV-Fuzzer* for the same number of simulations and record the critical scenarios that contain collisions between ego and NPC vehicles. These critical scenarios are classified into different types based on their logical patterns, i.e., the actions of the vehicles prior to each collision. (see §6.1 for classification results) We then measure their performance in terms of the following metrics:

- *The number of distinct types of critical scenarios, denoted as* #Types. This metric quantifies the total number of unique types of critical scenarios identified by different approaches.
- *The exposure rate of distinct types over the identified critical scenarios, denoted as* #TypeExposRate. This metric measures the proportion of critical scenarios that reveal a distinct type of collision. It is calculated as the ratio of the number of distinct types to the total number of critical scenarios identified.
- *the number of simulations required to detect the first type of critical scenarios, represented by* #SimForFirstType. This metric measures the number of simulations needed to identify the initial type of critical scenario.
- *the number of simulations required to detect all types of critical scenarios, denoted as* #SimForAllTypes. This metric quantifies the number of simulations needed to uncover all distinct types of critical scenarios.
- *the time cost for one scenario, denoted as* #TimeForOneScenario. This metric evaluates the efficiency of different approaches by calculating the time duration for a single scenario, including its generation, execution, and analysis.

To address RQ2, we conduct an ablation study to assess the usefulness of the extracted functional scenarios, especially the interactive pattern sequences, for the critical scenario generation process. For this purpose, we implement a variant of LeGEND, called LeGEND$^-$. In LeGEND$^-$, only one LLM is used to generate logical scenarios directly from accident reports, without extracting the functional scenarios. We run LeGEND$^-$ and LeGEND based on the same set of accident reports, and use the same number of simulations as the budget for searching concrete scenarios. We then compare the performances of them using the metrics in RQ1.

To address RQ3, we investigate the accuracy of LLM_1 for the extraction in Phase 1 and LLM_2 for the conversion in Phase 2. We answer RQ3 empirically by a user study, since it is difficult to assess the accuracy of the transformations. Specifically, we perform an online survey with 10 graduate students recruited from our department of computer science. Among the participants, six people have more than two years' research experience in the field of autonomous driving. In the study, since it takes one user about 5 minutes to finish the questions regarding one accident report, we randomly selected 5 accident reports from our database as test seeds. We then ask each participant to read each accident report and express their opinions on the statements regarding the extracted functional scenarios and converted logical scenarios. Participants are asked to rate their agreement on a 5-point Likert scale [23], ranging from "Strongly Disagree" to "Strongly Agree". In addition, we give participants an opportunity to provide supplementary feedback, such as brief textual comments, particularly for the instances on which their opinions are not strong.

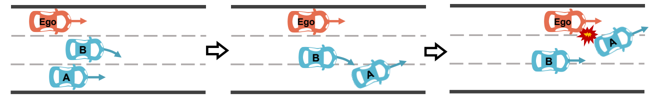


Figure 4: Example of a critical scenario derived from the accident report as illustrated in Fig. 3

Configurations LeGEND employs GPT4 [3] for both the extraction and conversion tasks, since this model is the state-of-the-art for a wide range of downstream tasks. We use the *gpt-4-0613* checkpoint with *max_token* of 1000 provided via the OpenAI API [1], and the temperatures are all set as 0.8. To ensure realism of scenarios, we select 20 accident reports from NHTSA database [29] as our initial seeds that cover different roads and number of vehicles. For the search algorithm, the crossover rate is set as 0.4, and the mutation rate is set as 0.5. Given that different test case templates vary in length (i.e., the number of statements in each test case), we set the population size as that length, following [17]. The maximum number of iterations for generation is set to 10. Each driving action is executed for a duration of 5 seconds following the trigger sequence. For RQ1 and RQ2, we run LeGEND and baseline approaches for the same number of 1400 simulations, and each execution lasts for more than 12 hours. For mitigating randomness, each experiment is repeated for 5 times and we take the average results.

Our experiments are conducted on a desktop with Ubuntu 20.04, 32GB of memory, an Intel Core i7-13700K CPU, and an NVIDIA RTX 3090 GPU. The simulation environment for the evaluation consists of Baidu Apollo 7.0 [5] (the target ADS) and LGSVL 2021.3 [33] (the software simulator). Apollo is an open-source and industry-grade modular ADS which supports a wide range of functionalities. We chose the San Francisco HD map provided by LGSVL, as this map includes various road structures that conform with the requirements about roads derived from the accident reports.

6 EVALUATION

6.1 RQ1: Can LeGEND improve diversity of scenarios, compared to baseline approaches?

6.1.1 Qualitative Analysis. During the execution, LeGEND identifies 96 critical scenarios that reveal safety violations. These violations are further categorized into 11 distinct types based on the action patterns that lead to each collision. These types of critical scenarios encompass a diverse range of road structures, number of vehicles, and driving actions. The distribution of critical scenarios across each type is shown in Fig. 5b. A comprehensive illustration of all types of critical scenarios is available on our website [4]. Due to space limitation, we select one type of critical scenario as an example to demonstrate that LeGEND can identify critical scenarios derived from real-world interactive patterns. The evolution of this scenario is explained below.

As shown in Fig. 4, the ego vehicle is traveling on a straight road with two NPC vehicles, denoted as *A* and *B*, driving in the adjacent two lanes. Suddenly, vehicle *B* attempts to change lanes into the lane that vehicle *A* drives and initiates a deceleration action. The ego vehicle detects the lane change of vehicle *B* and responds by slightly braking before continuing to drive straight. During vehicle

B's lane change, vehicle *A* decides to move into the lane that the ego drives. However, the ego vehicle continues driving forwards and fails to respond timely to vehicle *A*'s lane change, leading to a collision.

Note that the logical pattern of this critical scenario is derived from a real-world accident report as discussed in §3. The description of the report is shown in Fig. 3. During the testing process, LeGEND substitutes the "V3" vehicle with the ego vehicle controlled by the ADS to create a challenging driving environment.

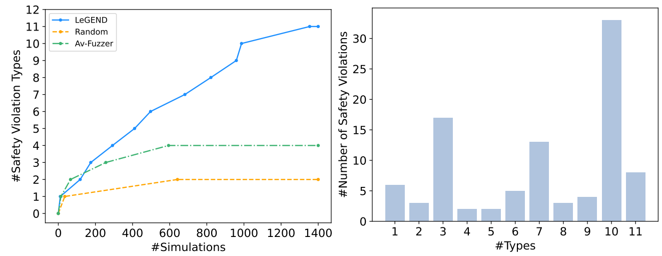
6.1.2 Quantitative Analysis. To further demonstrate the performance of LeGEND, we compare it with the two baseline techniques based on the five metrics as outlined in §5. Fig. 5a illustrates the growth in the number of critical scenario types discovered by the three approaches over simulations. We can observe that LeGEND outperforms the other methods since it manages to detect the highest number of distinct types of critical scenarios. Moreover, LeGEND continuously exposes new types of critical scenarios throughout the simulations. In comparison, *AV-Fuzzer* exhibits the fastest convergence but detects only five distinct types of critical scenarios. *Random* exhibits a similar convergence speed to *AV-Fuzzer* but identifies the fewest types of critical scenarios. Notably, all types of critical scenarios detected by both *AV-Fuzzer* and *Random* are exposed within the first 700 simulations.

LeGEND vs. *Random* Table 2 presents a comparative analysis between LeGEND and *Random*. Over the period of 1400 simulations, *Random* generates 64 critical scenarios that reveal potential system faults, with the first critical scenario identified during the 34-*th* simulation. It can identify 2 types of critical scenarios during the scenario generation process. All types of the critical scenarios are observed within the first 642 simulations.

Compared to *Random*, LeGEND generates a larger number of critical scenarios and uncovers a greater diversity of them. The type-exposure frequency #TypeExposRate of LeGEND is higher, which indicates its ability to generate diverse critical scenarios effectively. Additionally, the two types of critical scenarios identified by *Random* are also detected by LeGEND in the 118-*th* and 497-*th* simulation respectively. These comparative results demonstrate that LeGEND is more effective and efficient in generating critical scenarios with greater diversity in terms of types of scenarios.

LeGEND vs. *AV-Fuzzer* The comparison between LeGEND and *AV-Fuzzer* is also detailed in Table 2. During the 12-hour execution, *AV-Fuzzer* generates 297 critical scenarios with the first collision occurring in the 14-*th* simulation. The high number of identified critical scenarios by *AV-Fuzzer* can be attributed to the local search mechanism employed by this technique. However, despite the large number of critical scenarios, *AV-Fuzzer* is limited to exposing only 4 distinct types, resulting in a type exposure frequency of 1.3%. The four types of critical scenarios are all exposed by LeGEND in the 118-th, 411-*st*, 497-*th* and 822-*nd* simulation.

Compared to *AV-Fuzzer*, all four types of critical scenarios identified by *AV-Fuzzer* are also detected by LeGEND. Moreover, LeGEND exhibits a greater diversity of distinct critical scenarios during the critical scenario generation process. The type exposure rate for LeGEND is 11.5%, significantly higher than the 1.6% observed with *AV-Fuzzer*. Furthermore, LeGEND detects the first critical scenario more quickly and has a lower time cost per scenario, compared to



(a) Safety violation types discovered over simulations **(b) Number of safety violations of each type**

Figure 5: The trend of violation types over simulations and the distribution of safety violations over each type

Table 2: Comparison results with baselines

Metric	**Method**		
	LeGEND	***Random***	***AV-Fuzzer***
#Types	11	2	4
#TypeExposRate	11.5%	3.1%	1.3%
#SimForFirstType	11	35	14
#SimForAllTypes	1354	642	594
#TimeForOneScenario	32.1	46.2	46.6

AV-Fuzzer. These results demonstrate that LeGEND is more effective and efficient in exposing more diverse critical scenarios.

While LeGEND has additional time cost during the two-phase conversion process, such as the API calls to LLMs, we observe that its overall time cost is lower than that of the two baseline techniques, averaging 32.1 seconds per scenario. This efficiency can be attributed to two main factors: (1) the time required for API calls is minimal, compared to the duration of thousands of simulations; (2) as LeGEND pursues diversity, the test cases executed by LeGEND are often more lightweight than the cases executed by *AV-Fuzzer* that tries to search for more complicated cases for criticality. Usually, simpler test cases can also expose more serious system defects, since they are not supposed not be there. These factors contribute to the reduced time cost, enhancing the overall efficiency of LeGEND compared to the baseline techniques.

Answer to RQ1: Compared with existing techniques, LeGEND demonstrates an advantage in detecting critical scenarios with high diversity.

6.2 RQ2: Can interactive pattern sequences help to improve transformation precision?

To evaluate the effectiveness of the intermediate representation in our approach, we also implement a variant version of LeGEND, called LeGEND$^-$, in which the logical scenarios are generated directly from the accident report using a single LLM. We run LeGEND and LeGEND$^-$ for 1400 simulations respectively on the same set of accident reports. The comparison results are presented in Table 3.

Table 3: Comparison results of LeGEND and LeGEND$^-$ and cause distribution

Metric	**LeGEND**	LeGEND$^-$
#Types	11	2
#TypeExposRate	11.5%	7.1%
#SimForFirstType	11	81
#SimForAllTypes	1354	982
#TimeForOneScenario	32.1	31.9

#	C1	C2	C3	C4	Total
Number	1	4	2	2	9

During the execution, LeGEND$^-$ identifies 28 critical scenarios revealing potential system faults, with the first collision occurring in the 81-*st* simulation. These critical scenarios are categorized into 2 distinct types, achieving a type exposure rate of 7.1%. Both types were discovered within the first 982 simulations. In comparison, LeGEND is able to generate more critical scenarios and expose more diverse types of scenarios. Notably, all types of critical scenarios discovered by LeGEND$^-$ are also detected by LeGEND in the 118-*th* and 938-*th* simulations. This demonstrates that LeGEND not only identifies more critical scenarios but also uncovers a broader spectrum of distinct types, compared to LeGEND$^-$.

To investigate why LeGEND$^-$ exposes fewer types of critical scenarios than LeGEND, we further conduct a detailed comparison of the logical scenarios generated by the two approaches. Specifically, we focus on the accident reports that lead to the identification of a distinct type of critical scenarios by LeGEND but not by LeGEND$^-$. These accident reports, along with the logical scenarios generated by both approaches are then assigned to three authors of this paper as the assessors for the evaluation. The assessors are required to complete the following three stages for each accident report: (1) thoroughly read the original report and mark the necessary information related to the crash, such as interactions between vehicles; (2) compare the differences between the two logical scenarios represented in the same DSL; (3) record the main reasons for the differences in experimental results.

Finally, the assessment results are consolidated in a consensus meeting where the assessors compared their decisions. Any conflicting decisions are resolved through discussion. As a result, we conclude the following root causes:

- The logical scenario has an incorrect trigger sequence for the maneuvers of NPC vehicles, represented by **C1**;
- The logical scenario lacks necessary maneuvers of NPC vehicles that facilitate a violation, represented by **C2**;
- The logical scenario includes redundant maneuvers that could affect the substitution of the ego vehicle, represented by **C3**;
- The logical scenario contains parameter ranges that could not accurately reflect the interactions indicated from the original report, represented by **C4**.

The percentage of each type of cause is listed in Table 3. It can be observed that **C4** is the main cause for 44.4% (4 of 9) of the cases. Moreover, **C1**, **C2** and **C4** have a direct relation to the interactive actions between vehicles described in the original accident reports, accounting for 77.8% (7 of 9) of the causes. These findings emphasize the necessity of a semi-formal functional scenario, particularly the extraction of interactive pattern sequences, in improving the performance of LeGEND.

Answer to RQ2: The two-phase conversion process is essential. The intermediate representation plays a crucial role in ensuring transformation precision and exposing more diverse critical scenarios.

Table 4: Statements and report cases used for the user study

ID	Statement
1	The description of the road structure is correct.
2	The initial actions of vehicles are described accurately.
3	The interactive pattern sequence is extracted accurately.
4	The formal test case template matches the interactive pattern sequence.
5	The parameter ranges in the test case can accurately reflect the scenario.

#	Report ID	Short Description
Case 1	2005008586482	3-lane straight road, 3 vehicles involved, rear-end collision
Case 2	2006009501304	4-lane curved road, 2 vehicles involved, rear-end collision
Case 3	2005004496082	4-lane straight road, 3 vehicles involved, lane-change collision
Case 4	2006002585306	4-lane curved road, 2 vehicles involved, lane-change collision
Case 5	2006043699404	6-lane straight road, 3 vehicles involved, lane-change collision

6.3 RQ3: How accurate are LLM_1 and LLM_2 for the transformation tasks in LeGEND?

As mentioned in §5, we address RQ3 through a user study to investigate the accuracy of the two-stage conversion performed by LLMs. Table 4 lists the statements and the selected accident reports utilized in the study. We construct five statements specifically for evaluation purposes. The statements related to the intermediate representation cover aspects such as road structures (Statement #1), initial vehicle actions (Statement #2), and the interactive patterns (Statement #3). The statements related to logical scenarios focus on the correctness of the logical scenario template (Statement #4) and the parameter ranges (Statement #5). For the evaluation, we select 5 accident reports, with the main distinguishing features summarized in Table 4. The selected reports cover a wide range of road structures, numbers of vehicles and crash types, which facilitate a comprehensive assessment of the two-stage conversion process in LeGEND.

Fig. 6 presents the results of participants' responses across the five cases. The data indicates that participants generally agree that LeGEND accurately extracts the road structures and initial vehicle actions. Regarding the extracted interactive pattern sequences across the five cases, participants still agree but less strongly for

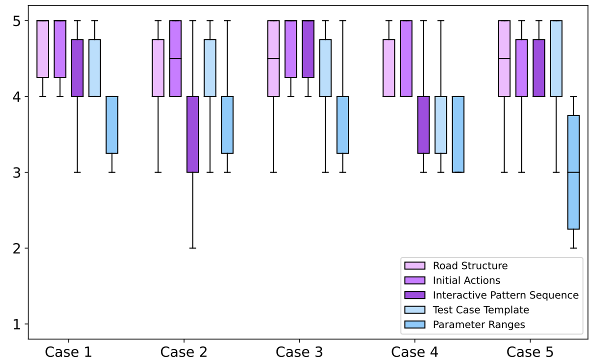


Figure 6: Results over different cases

*Case*2 and *Case*4. By the analysis of their feedback, we find that this discrepancy arises because the large language model (LLM) can generate new actions for vehicles when specific actions are not explicitly described in the accident report. This result aligns with the design choice of LeGEND, as LLMs are generative models characterized by their high creativity. Moreover, while the template conversion process is deemed accurate by participants, the parameter filling process is found to be less precise. This is attributed to the current limitations in the reasoning capabilities of LLMs.

Answer to RQ3: Overall, the transformations by LLMs are accurate. Although LLMs currently exhibit limitations in parameter filling for test cases, LeGEND mitigates this issue by adopting only a part of the parameters that have higher confidence.

6.4 Threats to Validity

Internal Validity The threats to internal validity could be the possibility of instrumentation effects stemming from faults within our approach LeGEND. These effects could introduce unintended biases or inaccuracies in the generated scenarios and evaluation results. To mitigate these threats, we conducted thorough testing of the implementation and inspected the intermediate results during execution. To avoid bias in the user survey, we anonymized it and assigned tasks to participants randomly.

External Validity The threats to external validity could be the selected accident reports. We only selected a small number of police reports from the NHTSA database to conduct the experiments; however, we ensured that the selected reports cover different crash types, road geometries, number of cars, and car movements. Since there exists perception delay of the software simulator, i.e., LGSVL, we mitigate this by bypassing the perception module and sending the ground truth of the perception to the ADS. Another threat is the generalizability of our approach to other types of LLMs and ADSs under test. To mitigate this threat, we selected GPT-4 [3] for the extraction and conversion tasks, as it was one of the most advanced and powerful LLMs available as of the time of performing our experiments. For the ADS under test, we adopted Baidu Apollo [5], an open-source, industrial-grade ADS that is one of the most popular platforms and has been extensively studied [9, 16, 21, 46] in literature of autonomous driving. Therefore, we believe the selected LLM and ADS under test are representative for our approach. A final threat to external validity is the eligibility of the participants of our user study. To mitigate this issue, we involved graduate students who have been working extensively in this domain for multiple years, and moreover, these participants were required to read the accident reports thoroughly and rate their agreement with the corresponding questions carefully, therefore, they are knowledgeable to the problem under investigation and able to make precise and reliable assessments.

7 RELATED WORK

Large Language Models have been widely adopted due to its extraordinary performances in different downstream tasks [20, 27, 30]. Currently, there are two main techniques that employ LLMs for domain-specific tasks, namely, prompt engineering [24] and fine-tuning [31]. Prompt engineering, such as few-shot learning [6], works by providing a description of the task with a few demonstrations to directly elicit the desired outputs. In contrast, fine-tuning involves training a pre-trained model on a task-specific dataset to adapt it to particular tasks, e.g., code generation [34]. However, since fine-tuning an LLM is time-consuming and expensive, prompting strategies have been increasingly adopted [45]. In this work, our LLM-related tasks are accomplished via prompt engineering.

Module-level ADS testing targets the evaluation of individual components within an ADS, including the perception, planning, and control modules. Testing of the perception module involves generating adversarial inputs to challenge the machine learning models dedicated to various perception tasks, such as object detection [8, 36], semantic segmentation [42], and traffic sign recognition [22]. For example, Cao et al. [8] propose an optimization-based attack method aimed at generating adversarial 3D objects to deceive both camera-based and LiDAR-based object detectors within ADS perception systems. Meanwhile, testing of the planning and control modules primarily focuses on assessing the safety of planned trajectories [18, 19] or verifying the correctness of control decision [12]. These evaluations often rely on dedicated path planners and controllers [12, 18, 19]. However, existing module-level testing approaches tend to overlook the potential issues arising from the complex collaborations among different modules.

System-level ADS testing consider the interactions and dependencies among its different modules, and often evaluate an ADS by system simulation. To effectively find critical scenarios that can reveal the system faults, various approaches have been proposed including search-based testing [9, 14, 16, 21, 32, 39, 40, 46, 47], metamorphic testing [15, 49] and formal methods [37, 44, 48]. Among these methodologies, search-based techniques are prominent for detecting critical scenarios in a huge input space. For instance, Tian et al. [40] encode atomic driving behaviors as motif patterns, and applies a multi-objective genetic algorithm to generate diverse test scenarios. Zhong et.al [46] employ a neural network to guide the process of seed selection, thereby accelerating evolutionary search. There is a line of work [13, 35, 43] that aim to enhance realism of test cases. For example, Gambi et al. [13] propose *AC3R*, which can recover vehicle collisions from public accident reports based on natural language processing techniques. Zhang et al. [43] employ a segmentation model to extract information for reconstructing scenes derived from actual accident data. However, most of these studies [13, 16, 21, 37, 39, 40, 43, 44, 47, 48] focus on generating or reconstructing concrete scenarios without considering functional levels, so they cannot control scenario diversity at a high level.

8 CONCLUSION AND FUTURE WORK

Detecting diverse safety violations is crucial for ADS testing, because it can expose syste defects in different aspects and avoid redundant search space exploration. In this paper, we propose LeGEND that features a top-down style of scenario generation for testing of the ADS that starts with functional scenarios and then steps downwards. Specifically, LeGEND employs two LLMs to transform functional scenarios to formal logical scenarios, and then searches with the logical scenarios for critical scenarios. As future work, we plan to try functional scenarios from other sources to further study the performances of LLMs in ADS testing.

REFERENCES

[1] 2023. Models - GPT-4. https://platform.openai.com/docs/models/gpt-4.

[2] Sherif M Abuelenin and Adel Y Abul-Magd. 2015. Effect of minimum headway distance on connectivity of VANETs. *AEU-International Journal of Electronics and Communications* 69, 5 (2015), 867–871.

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[4] Anonymous Authors. 2024. LeGEND. https://sites.google.com/view/legend4adstesting.

[5] Baidu. [n. d.]. An open autonomous driving platform. https://github.com/ApolloAuto/apollo.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[7] Alessandro Calò, Paolo Arcaini, Shaukat Ali, Florian Hauer, and Fuyuki Ishikawa. 2020. Generating avoidable collision scenarios for testing autonomous driving systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 375–386.

[8] Yulong Cao, Ningfei Wang, Chaowei Xiao, Dawei Yang, Jin Fang, Ruigang Yang, Qi Alfred Chen, Mingyan Liu, and Bo Li. 2021. Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 176–194.

[9] 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*. 488–500.

[10] Tesla Deaths. [n. d.]. A record of Tesla-related fatalities. https://www.tesladeaths.com/.

[11] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[12] Adel Djoudi, Loic Coquelin, and Rémi Regnier. 2020. A simulation-based framework for functional testing of automated driving controllers. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–6.

[13] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 257–267.

[14] 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*. 318–328.

[15] 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*. 380–385.

[16] Yuqi Huai, Yuntianyi Chen, Sumaya Almanee, Tuan Ngo, Xiang Liao, Ziwen Wan, Qi Alfred Chen, and Joshua Garcia. 2023. Doppelgänger test generation for revealing bugs in autonomous driving software. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2591–2603.

[17] Russell Lande, George F Barrowclough, et al. 1987. Effective population size, genetic variation, and their use in population management. *Viable populations for conservation* 87 (1987), 87–124.

[18] Thomas Laurent, Paolo Arcaini, Fuyuki Ishikawa, and Anthony Ventresque. 2019. A mutation-based approach for assessing weight coverage of a path planner. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 94–101.

[19] Thomas Laurent, Paolo Arcaini, Fuyuki Ishikawa, and Anthony Ventresque. 2020. Achieving Weight Coverage for an Autonomous Driving System with Search-based Test Generation. In *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 93–102.

[20] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K Lahiri, and Siddhartha Sen. 2023. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 919–931.

[21] 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)*. IEEE, 25–36.

[22] Yujie Li, Xing Xu, Jinhui Xiao, Siyuan Li, and Heng Tao Shen. 2020. Adaptive square attack: Fooling autonomous cars with adversarial traffic signs. *IEEE Internet of Things Journal* 8, 8 (2020), 6337–6347.

[23] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[24] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[25] Till Menzel, Gerrit Bagschik, Leon Isensee, Andre Schomburg, and Markus Maurer. 2019. From functional to logical scenarios: Detailing a keyword-based scenario description for execution in a simulation environment. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2383–2390.

[26] Till Menzel, Gerrit Bagschik, and Markus Maurer. 2018. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1821–1827.

[27] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.

[28] National Highway Traffic Safety Administration (NHTSA). [n. d.]. Case ID: 2005004496082. https://crashviewer.nhtsa.dot.gov/nass-NMVCCS/CaseForm.aspx?xsl=main.xsl&CaseID=2005004496082.

[29] National Highway Traffic Safety Administration (NHTSA). [n. d.]. National motor vehicle crash causation survey. https://crashviewer.nhtsa.dot.gov/LegacyNMVCCS/Search.

[30] Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. 2023. Understanding the effectiveness of large language models in code translation. *arXiv preprint arXiv:2308.03109* (2023).

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[32] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 876–888.

[33] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*. IEEE, 1–6.

[34] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[35] Christoph Sippl, Benedikt Schwab, Peter Kielar, and Anatoli Djanatliev. 2018. Distributed real-time traffic simulation for autonomous vehicle testing in urban environments. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2562–2567.

[36] Jiachen Sun, Yulong Cao, Qi Alfred Chen, and Z Morley Mao. 2020. Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 877–894.

[37] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. 2022. LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[38] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, et al. 2023. A survey on

automated driving system testing: Landscapes and trends. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–62.

[39] Shuncheng Tang, Zhenya Zhang, Jixiang Zhou, Yuan Zhou, Yan-Fu Li, and Yinxing Xue. 2023. EvoScenario: Integrating Road Structures into Critical Scenario Generation for Autonomous Driving System Testing. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 309–320.

[40] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 94–106.

[41] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. 2015. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th international conference on intelligent transportation systems*. IEEE, 982–988.

[42] Xing Xu, Jingran Zhang, Yujie Li, Yichuan Wang, Yang Yang, and Heng Tao Shen. 2020. Adversarial Attack Against Urban Scene Segmentation for Autonomous Vehicles. *IEEE Transactions on Industrial Informatics* 17, 6 (2020), 4117–4126.

[43] Xudong Zhang and Yan Cai. 2023. Building critical testing scenarios for autonomous driving from real accidents. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 462–474.

[44] Xiaodong Zhang, Wei Zhao, Yang Sun, Jun Sun, Yulong Shen, Xuewen Dong, and Zijiang Yang. 2023. Testing automated driving systems by breaking many laws efficiently. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 942–953.

[45] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. 2023. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923* (2023).

[46] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2022. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* (2022).

[47] Jixiang Zhou, Shuncheng Tang, Yan Guo, Yan-Fu Li, and Yinxing Xue. 2023. From Collision to Verdict: Responsibility Attribution for Autonomous Driving Systems Testing. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 321–332.

[48] Yuan Zhou, Yang Sun, Yun Tang, Yuqi Chen, Jun Sun, Christopher M Poskitt, Yang Liu, and Zijiang Yang. 2023. Specification-based Autonomous Driving System Testing. *IEEE Transactions on Software Engineering* (2023).

[49] Zhi Quan Zhou and Liqun Sun. 2019. Metamorphic testing of driverless cars. *Commun. ACM* 62, 3 (2019), 61–67.