DiaVio: LLM-Empowered Diagnosis of Safety Violations in ADS Simulation Testing

You Lu*
School of Computer Science
Fudan University
Shanghai, China

Yifan Tian*
School of Computer Science
Fudan University
Shanghai, China

Yuyang Bi*
School of Computer Science
Fudan University
Shanghai, China

Bihuan Chen*†
School of Computer Science
Fudan University
Shanghai, China

Xin Peng*
School of Computer Science
Fudan University
Shanghai, China

ABSTRACT

Simulation testing has been widely adopted by leading companies to ensure the safety of autonomous driving systems (ADSs). A number of scenario-based testing approaches have been developed to generate diverse driving scenarios for simulation testing, and demonstrated to be capable of finding safety violations. However, there is no automated way to diagnose whether these violations are caused by the ADS under test and which category these violations belong to. As a result, great effort is required to manually diagnose violations.

To bridge this gap, we propose DiaVio to automatically diagnose safety violations in simulation testing by leveraging large language models (LLMs). It is built on top of a new domain specific language (DSL) of crash to align real-world accident reports described in natural language and violation scenarios in simulation testing. DiaVio fine-tunes a base LLM with real-world accident reports to learn diagnosis capability, and uses the fine-tuned LLM to diagnose violation scenarios in simulation testing. Our evaluation has demonstrated the effectiveness and efficiency of DiaVio in violation diagnosis.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Automated Driving System, Scenario-based Testing, Large Language Models, Violation Diagnosis

ACM Reference Format:
You Lu, Yifan Tian, Yuyang Bi, Bihuan Chen, and Xin Peng. 2024. DiaVio: LLM-Empowered Diagnosis of Safety Violations in ADS Simulation Testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650212.3652135

*Y. Lu, Y. Tian, Y. Bi, B. Chen, and X. Peng are also with the Shanghai Key Laboratory of Data Science.
†Bihuan Chen is the corresponding author.

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3652135

1 INTRODUCTION

In recent decades, there has been a significant escalation in both academic and industrial commitment towards the development of autonomous driving systems (ADSs), which have notable implications for automotive transportation and societal benefits [52]. Unfortunately, current ADSs from leading companies like Tesla, Waymo, and Uber are still vulnerable to corner cases and exhibit incorrect behaviors, due to the extremely complicated and diverse real-world driving environments. Such erroneous behaviors in ADSs can result in severe consequences and substantial losses, as exemplified by numerous documented traffic incidents [8, 33, 45].

Consequently, leading companies have employed on-road testing to ensure the reliability of ADSs. However, to demonstrate with 95% confidence that autonomous vehicles are 20% safer than human drivers, autonomous vehicles would have to be driven more than 11 billion miles [30]. It is expensive for on-road testing to achieve this goal, and it is also impossible for on-road testing to test corner cases or dangerous situations. To this end, simulation testing has been widely adopted by these leading companies [29, 32]. Recently, many scenario-based testing approaches [1, 2, 11, 14, 15, 22, 24, 27, 34–36, 42–44, 56, 58, 59, 63, 64] have been proposed to generate diverse driving scenarios for simulation testing.

These scenario-based testing approaches have been demonstrated to be capable of finding safety violations. However, these violations do not necessarily reveal a bug in the ADS under test because the EGO vehicle (i.e., the vehicle controlled by the ADS under test) may not bear the responsibility of these violations. For example, NPC vehicles (i.e., other vehicles except the EGO one) may not obey traffic signals and collide with the EGO vehicle, making NPC vehicles take the responsibility of the violation. This is also evidenced by a recent study [28], where 1,109 crash scenarios are automatically generated in 240 hours. After manual diagnosis, all these violations are solely the responsibility of NPC vehicles, revealing zero bug for the ADS. Therefore, *it becomes important to automatically diagnose the violations detected by existing scenario-based testing approaches.* To the best of our knowledge, none of the existing scenario-based testing approaches are equipped with the violation diagnosis capability.

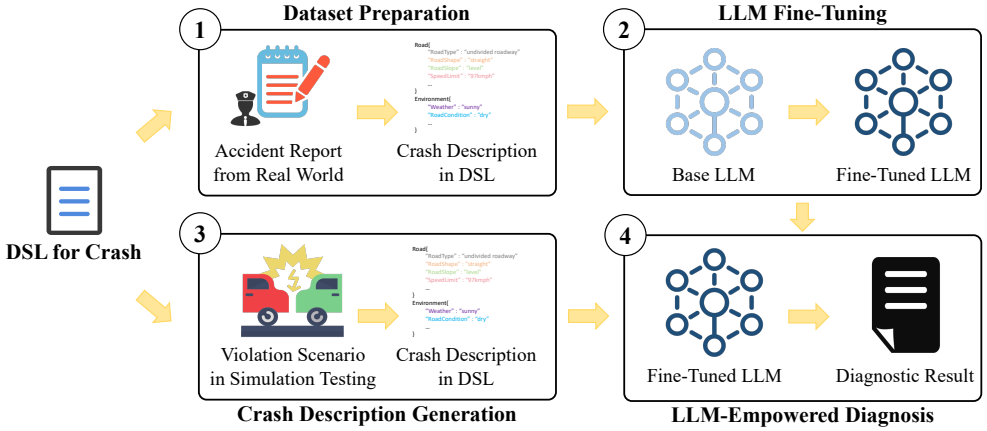


Figure 1: Approach Overview of DiaVio

To bridge this gap, we propose DiaVio, a novel approach that leverages large language models (LLMs) for automatically diagnosing safety violations in ADS simulation testing. We design the diagnosis capability from two aspects. The first is *liability determination* capability, which determines whether the EGO vehicle or the NPC vehicle bears the responsibility for a violation. This can reduce the cost of manually eliminating violations caused by NPC vehicles. The second is *crash classification* capability, which classifies the crash in a violation into a specific category (e.g., backover collision). This can group similar violations together and ease the bug diagnosis.

DiaVio is built upon a new domain specification language (DSL) for describing crashes. Our crash DSL aims to align real-world accident reports described in natural language and violation scenarios in simulation testing. In the training phase of DiaVio, it parses real-world accident reports (which not only provide crash details but also diagnostic result) into crash descriptions in the syntax of our crash DSL, and fine-tunes a base LLM based on these crash descriptions to learn the diagnosis capability. In the inference phase of DiaVio, it generates a crash description, following our crash DSL, for each violation scenario identified in simulation testing, and utilizes our fine-tuned LLM to diagnosis the violation.

We have conducted large-scale experiments to evaluate the effectiveness and efficiency of DiaVio. First, we fine-tune three base LLMs with the accident reports in NMVCCS [5] and CIREN [3] datasets. The best of our fine-tuned LLMs has an accuracy of 87.06% and 85.44% in liability determination and crash classification for accident reports, respectively. Second, we use the best of our fine-tuned LLMs to diagnose the violation scenarios generated by two scenario-based testing approaches, i.e., AV-Fuzzer [36] and DriveFuzz [34]. AV-Fuzzer and DriveFuzz respectively generate 192 and 19 violation scenarios. Two of the authors take three days to manually diagnose these violations to establish the ground truth. The best of our fine-tuned LLMs takes two hours and achieves an accuracy of 93.84% and 96.21% in liability determination and crash classification for these violations, which significantly reduces the manual cost. Finally, we assess the quality of accident report parsing and the consistency of our LLM-empowered violation diagnosis, which are satisfactory.

The main contributions of our work are summarized as follows.

- We proposed a crash DSL to align real-world traffic accident reports and violation scenarios in simulation testing.
- We developed DiaVio to leverage LLM for the automated diagnosis of safety violations in ADS simulation testing.
- We conducted experiments with two scenario-based testing approaches to demonstrate DiaVio's effectiveness and efficiency.

2 METHODOLOGY

We propose and implement DiaVio to automatically diagnose safety violations identified by ADS simulation testing techniques. The approach overview of DiaVio is presented in Fig. 1. The overall idea of DiaVio is to learn a diagnosis model from real-world traffic accident reports, and use the learned model to diagnose violation scenarios in simulation testing. Here, the challenge is to bridge the semantic gap between accident reports described in natural language and violation scenarios in simulation testing. To this end, we propose a domain-specific language (DSL) to unifiedly describe crashes in both accident reports and violation scenarios (see Sec. 2.1).

Given the crash DSL, DiaVio consists of two phases. In the first training phase, DiaVio prepares the dataset (i.e., step ① in Fig. 1) by transforming each accident report into a crash description that follows the DSL syntax (see Sec. 2.2). Then, DiaVio fine-tunes a base large language model (LLM) (i.e., step ② in Fig. 1) for two diagnosis tasks (see Sec. 2.3). The task of *liability determination* is to decide which vehicle bears the responsibility for a crash occurrence, and explain the determination reason. The task of *crash classification* is to classify a crash into a specific category (e.g., backover collision or frontal collision), and explain the classification reason.

In the second inference phase, DiaVio generates a crash description (i.e., step ③ in Fig. 1), which follows the DSL syntax, for each violation scenario identified in ADS simulation testing (see Sec. 2.4). Then, DiaVio diagnoses the violation scenario (i.e., step ④ in Fig. 1) by feeding its corresponding crash description into the fine-tuned LLM (see Sec. 2.5). The diagnostic results are reported to ADS developers to improve the ADS in a cost-efficient way.

2.1 Domain Specification Language for Crash

We propose a DSL to describe crashes. The DSL serves as an intermediate representation to align crashes in accident reports in natural language and crashes in violation scenarios in simulation testing.

```
<Crash> ::= <Road>; <Environment>; <Obstacles>;
            <Vehicles>; <Diagnosis>;

<Road> ::= <RoadType>; <Lanes>; <RoadShape>;
           <RoadSlope>; <SpeedLimit>;
<RoadType> ::= intersection | cityStreet | ruralRoad | ...
<Lanes> ::= ∅ | <Lane>; <Lanes>;
<Lane> ::= [laneIdentifier] | unmentioned
<RoadShape> ::= straight | curved | inclined | ...
<RoadSlope> ::= flat | uphill | downhill | ...
<SpeedLimit> ::= [numerical] | unmentioned

<Environment> ::= <Weather>; <RoadCondition>; <TrafficSignals>;
<Weather> ::= sunny | rainy | foggy | ...
<RoadCondition> ::= dry | wet | icy | ...
<TrafficSignals > ::= red | yellow | green | unmentioned

<Obstacles> ::= ∅ | <Obstacle>; <Obstacles>;
<Obstacle> ::= <Type>; <Location>;
<Type> ::= non-motorized vehicle | roadblock | ...
<Location> ::= [coordinate] | unmentioned

<Vehicles> ::= ∅ | <Vehicle>; <Vehicles>;
<Vehicle> ::= <ImpactSide>; <MovingOnWhichWay>;
              <LocationAfterCrash>; <Behavior>;
<ImpactSide> ::= front | rear | left | right | ...
<MovingOnWhichWay> ::= <Lane>
<LocationAfterCrash> ::= [coordinate] | unmentioned
<Behavior> ::= <Direction>; <VehicleAction>; <TravelSpeed>;
               <WhetherToBrake>; <AttemptedLaneCrossing>;
               <IsAgainstRules>;
<Direction> ::= north | south | east | west | unmentioned
<VehicleAction> ::= accelerating | decelerating | changingLane | ...
<TravelSpeed> ::= [numerical] | unmentioned
<WhetherToBrake> ::= yes | no | unmentioned
<AttemptedLaneCrossing> ::= left-change | right-change | no | unmentioned
<IsAgainstRules> ::= [rule] | no | unmentioned

<Diagnosis> ::= <ResponsibleParty>; <ReasonForLiability>;
                <CrashCategory>; <ReasonForCategory>;
<ResponsibleParty> ::= <Vehicle>
<ReasonForLiability> ::= [reason]
<CrashCategory> ::= Rear-End | Frontal | Front-to-Side | ...
<ReasonForCategory> ::= [reason]
```

Figure 2: The Syntax of Crash DSL

Existing DSLs (e.g., [21, 53]) are designed for testing scenario generation, and thus have complex syntax. Differently, our DSL is specifically designed for describing crashes that are not supported in existing DSLs, and thus have simple syntax. We define semantic elements derived from real-world accident reports to describe crashes.

Fig. 2 presents the syntax of our proposed DSL. A crash consists of five components, i.e., ***Road***, ***Environment***, ***Obstacles***, ***Vehicles*** and ***Diagnosis***. Each component is composed of subcomponents and atomic elements, which is elaborated as follows.

- ***Road*** represents the geographical context of a crash, including the type of road (*RoadType*), the lanes on the road (*Lanes*) whose identifiers are numbered from the left-most lane to the right-most lane, the shape of the road (*RoadShape*), the gradient of the road (*RoadSlope*), and the speed limit of the road (*SpeedLimit*).
- ***Environment*** characterizes the weather conditions (*Weather*) and the road surface conditions (*RoadCondition*) when a crash occurs. If the crash occurs at an intersection, the status of the traffic lights (*TrafficSignals*) at the time of the crash is also included.
- ***Obstacles*** describe whether there are obstacles on the road when a crash occurs, including the type of each obstacle (*Type*) (e.g., a non-motorized vehicles parked on the roadside, or a roadblock), and its coordinate on the road (*Location*).
- ***Vehicles*** indicate the status of each dynamic vehicle involved in a crash, including which part of the vehicle is hit (*ImpactSide*), which lane the vehicle is running (*MovingOnWhichWay*), the location where the vehicle stops after the crash (*LocationAfterCrash*), and the vehicle's behavior before the crash (*Behavior*). Specifically, vehicle behavior is further composed of six subcomponents. In detail, *Direction* describes the direction in which the vehicle is running. *VehicleAction* describes the action the vehicle is taking, which can be one of the ten types of actions [49, 50], including decelerating, accelerating, starting, passing, parking, turning left, turning right, backing up, changing lanes, and merging. *TravelSpeed* indicates the speed of the vehicle for a period of time before the crash. *WhetherToBrake* indicates whether the vehicle applies the brakes. *AttemptedLaneCrossing* indicates whether the vehicle attempts to change lanes and how it changes lanes. *IsAgainstRules* describes the violated traffic rules (e.g., running red lights).
- ***Diagnosis*** specifies the diagnostic result of a crash, including which vehicle is responsible for the crash (*ResponsibleParty*) and the reason for the vehicle to take the liability (*ReasonForLiability*), and the category of the crash (*CrashCategory*) and the reason for such a category classification (*ReasonForCategory*). Najm et al. [49, 50] have classified crashes into 37 categories. Based on their categories, we consolidate minor categories, resulting in six major categories, i.e., single-vehicle crash, backover collision, rear-end collision, frontal collision, front-to-side collision, and others.

For each subcomponent, we derive a list of atomic elements to represent common properties about crashes. For example, the atomic elements for *Weather* can be sunny, rainy, foggy, etc. If a subcomponent is not involved in a crash, it is set to unmentioned. Due to space limitation, Fig. 2 only shows part of the atomic elements, and the full list is available at our replication site.

2.2 Dataset Preparation

We propose to use real-world accident reports, which provide not only the crash detail but also the diagnostic result, as the dataset to learn a diagnosis model. As accident reports are described in natural language, we prepare the dataset by first collecting accident reports and then converting them into crash descriptions in the syntax of our crash DSL. We elaborate each step as follows.

Accident Report Collection. National Highway Traffic Safety Administration (NHTSA) [4] is an organization to collect information about vehicle crashes so as to aid in the development and evaluation of vehicles and highway safety. It provides real-world traffic accident report datasets such as National Motor Vehicle Crash Causation Survey (NMVCCS) dataset [5] and Crash Injury Research Engineering Network (CIREN) dataset [3]. We crawl these two datasets from its website. The accident reports in these two datasets typically describe the crashes in natural language, including road condition, weather condition, obstacles and vehicle behaviors. Besides, each accident report in the NMVCCS and CIREN dataset quantifies factors that contribute to the crash, while each accident report in the NMVCCS dataset further provides an analysis of the crash category.

Data Format Conversion. We use GPT-4 [51], which has strong capability in natural language understanding and text extraction, to

Table 1: An Example of Interacting with GPT-4 with Prompts

Role Setting:
You are an assitant of text extraction. I'll give you a piece of text and a series of questions. You need to answer me with key-value pairs based on the text. If the answer is not included in the text, you need to reply with "unmentioned".
Prompt:
{*the natural language description of an accident report*}
…
What's the speed limit of the road?
What's the weather like?
…
GPT-4 Response:
…
Speed limit of the road: "89 KPH (55 MPH)"
Weather Conditions: "clear"
…
Prompt:
Find out an answer in {*the content of the crash DSL*} with the closest meaning to your answer. If there is no matching answer, keep your original answer.
GPT-4 Response:
…
"SpeedLimit": "89kmp"
"Weather": "sunny"
…

obtain structured description of crashes in accident reports. Based on our predefined DSL in Fig. 2, we formulate a series of questions to interact with GPT-4. As illustrated in Table 1, we first set the role of GPT-4 primarily as a tool for extracting crash information. Then, we feed an accident report along with a list of questions to GPT-4, thereby extracting crucial information necessary for crash diagnosis. Finally, we instruct GPT-4 to align the extracted information with the syntax of our proposed crash DSL. This last step is crucial to ensure that each piece of information from the accident report aligns precisely with our crash DSL, avoiding misinterpretation due to semantically similar words. We guide GPT-4 to carefully select values from the DSL that most closely match the intended meaning in the accident description, thereby updating the extracted information for a coherent and accurate structured data format.

2.3 LLM Fine-Tuning

Base LLMs pre-trained with large-scale text data may already possess strong contextual understanding and generation capabilities. We hope to enhance the performance of base LLMs on our specific tasks of crash diagnosis through fine-tuning. Specifically, we construct fine-tuning dataset by the 3-tuple $\langle instruction, input, output \rangle$ based on our prepared dataset in Sec. 2.2. The *instruction* part sets the model's role as an assistant of two diagnosis tasks, i.e., liability determination and crash classification. The *input* part includes the ***Road***, ***Environment***, ***Obstacles*** and ***Vehicles*** components in the crash descriptions for accident reports. The *output* part corresponds to the ***Diagnosis*** component. We fine-tune a base LLM so that it infers the output based on the instruction and input, continuously improving the fine-tuned LLM's performance on the diagnosis tasks.

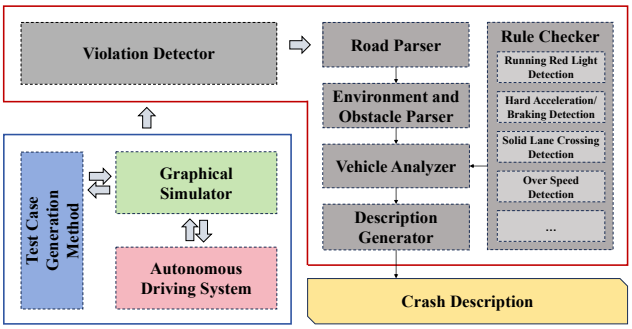


Figure 3: The Architecture of Crash Description Generation

Full fine-tuning that retrains all the parameters is very expensive and challenging for base LLMs with billions of parameters. Therefore, we adopt LoRA (Low-Rank Adaptation) [25], which reduces the number of trainable parameters by freezing the pre-trained model weights and injecting trainable rank decomposition matrices into each layer of the model, to fine-tune base LLMs using the dataset prepared in Sec. 2.2. According to Raschka's work [54], we set *lora_r* to 16 and *lora_alpha* to 32 for reduced memory requirement, less time consumption and better performance. Besides, we use the AdamW optimizer [40] because it is a common choice for LLM fine-tuning. In addition, we set *batch size* to 128, *epochs* to 4, and *learning rate* to $3e^{-4}$. Notice that we use different base LLMs, which will be introduced in Sec. 3.1.

2.4 Crash Description Generation

We develop a crash description generator to parse each violation scenario detected in simulation testing to a crash description in the syntax of our crash DSL. Fig. 3 presents the architecture of our crash description generator. The modules in the red box are developed by us, while the modules in the blue box are essentially from the existing simulation testing approaches. These existing approaches typically consist of three parts, i.e., a graphical simulator, an ADS under testing, and a test case generation method. Testers usually bridge the ADS with the simulator, and adopt a test case generation method, which changes scenario elements such as weather and behavior of NPC actors, to generate violation scenarios. Once a violation scenario occurs in simulation testing, our crash description generator, consisting of the following six key modules, starts to work.

- **Violation Detector.** We monitor each run of the simulation, and call subsequent modules to generate a description of the violation scenario when a crash between the EGO vehicle and the NPC vehicles is detected. This crash detection is implemented by directly calling the simulator's built-in Python API.
- **Road Parser.** We parse the HD map file in the *base_map.bin* or *OpenDRIVE* format from the simulator to extract the information about the road where the crash occurs. Specifically, the HD map file contains the information about the full road. It directly specifies the speed limit, road type, and road shape of each lane, and also includes coordinate information and unique identifiers for lanes, roads, junctions, and traffic signals. Using this information, we construct a dictionary to store the binding relationships between lanes and roads, between lanes and junctions, and between traffic lights and lanes. As we only need the information

about a specific part of the road where the crash occurs, we first obtain the EGO vehicle's position when the crash occurs through the simulator's Python API, and then locate the specific part of the road according to the EGO vehicle's position, and finally extract information of *RoadType*, *Lane*, *RoadShape*, *RoadSlope* and *SpeedLimit* for the specific part of the road.

- **Environment and Obstacle Parser.** We parse the environment condition of the crash and get the information of the obstacles from the simulator. Specifically, we first extract information of *Weather* and *RoadCondition* through the Python API provided by the simulator. Second, from the dictionary constructed by **Road Parser**, we query the status of the traffic light that is bound to the lane where the EGO vehicle's position locates, and obtain information of *TrafficSignals*. Finally, we obtain information of obstacles on the lane where the EGO vehicle travels, including their types and their coordinates, through the simulator's Python API. In this way, we extract information of *Type* and *Location*.
- **Vehicle Analyzer.** We get detailed information of vehicles involved in the crash and analyze their behaviors. First, we get the status of involved EGO and NPC vehicles from the simulator using Python API, including their speed, position, direction, angular velocity, and rotation (the angular difference between the vehicle's heading direction and the y-axis of the simulator's world map). In this way, we obtain information of *TravelSpeed*, *Direction* and *LocationAfterCrash*. Second, using the dictionary constructed by **Road Parser**, we query the lane where the vehicles run from the vehicles' position trajectory before the crash. Thus, we obtain information of *MovingOnWhichWay*. Third, based on the positions of the two colliding vehicles and their rotations, we determine the relative spatial relationship between the two vehicles, and then determine the impact side of each vehicle (i.e., *ImpactSide*). Finally, by measuring the changes of a vehicle's speed and its steering angle before the crash, we determine the vehicle's action, braking behavior, and tendency to cross lanes. In this way, we obtain information of *VehicleAction*, *WhetherToBrake* and *AttemptedLaneCrossing*. Besides, to determine whether a vehicle violates traffic rules (i.e., *IsAgainstRules*), we use **Rule Checker**, which will be introduced next.
- **Rule Checker.** We monitor the vehicle's driving behavior to determine whether the vehicle is against certain rules. Currently, we support the checking of two type of rules. One is to check whether vehicle's behavior complies with traffic regulations, including whether a vehicle has run a red light, crossed solid lane markings, or exceeded the speed limit. The other is to check whether vehicle's behavior is reasonable, such as engaged in hard acceleration or braking. The checker can be extended to support other rules.
  - **Running Red Light Detection.** We check all timestamps when the traffic lights are red during each simulation run. If a vehicle is found to be beyond the stop line at a junction with a non-zero speed, we consider it as a violation.
  - **Solid Lane Crossing Detection.** We retain a vehicle's trajectory information to match it with the HD map information. If the vehicle's trajectory crosses solid lane markings on the HD map, we consider it as a violation.
  - **Over Speed Detection.** We check a vehicle's speed during simulation. If there is a continuous period of time (e.g., 10 frames) where the vehicle's speed exceeds the lane's speed limit, we consider it as a violation.
  - **Hard Acceleration/Braking Detection.** We calculate the hard acceleration/braking indicator $K_{ab} = a/g$ [12] where a represents the acceleration of a vehicle and g represents the gravitational constant. We use ± 0.6 as a decision boundary for K_{ab}, following the prior work [34]. Specifically, we consider a vehicle to conduct a hard acceleration if $K_{ab} \geq 0.6$ or conduct a hard braking if $K_{ab} \leq -0.6$.
- **Description Generator.** This module is to generate a crash description aligning with our crash DSL. To this end, based on the syntax of our crash DSL, we create a description template that serves as a framework for describing the crash. This template is populated with data extracted from the preceding modules, ensuring a comprehensive characterization of the crash.

2.5 LLM-Empowered Diagnosis

After generating the crash description for a detected violation scenario (where the ***Diagnosis*** component is missing), we use the fine-tuned LLM in Sec 2.3 to diagnose the violation (i.e., to infer the ***Diagnosis*** component). Specifically, similar to fine-tuning, we set the fine-tuned LLM's role as an assistant of the two diagnosis tasks (i.e., we set the instruction), feed the generated crash description as the input to the fine-tuned LLM, and obtain its inferred output.

We observe that the instruction and input for each training data in Sec. 2.3 is approximately 520 tokens, and the output is around 120 tokens. Hence, we configure the maximum context tokens to 1,024. This extended token limit accommodates the processing of longer and more complex crashes, ensuring that input and output will not be truncated. Additionally, we set the *max_new_tokens* parameter to 200, which affects the length of the model's output. This allows the model to generate focused and relevant output, particularly in identifying key factors of a crash. Besides, *temperature* and *top_p* affect the consistency of outputs of the fine-tuned LLM. We choose to change *temperature* instead of changing both of them, following the practice in prior work [60]. Higher *temperature* encourages the model to take more risk, making the output creative, but sometimes it can also be hurtful, making the output too diverge. Here, we set *temperature* to 0.5, ensuring that the model maintains consistency in diagnosis while giving creative inferences.

3 EVALUATION

To evaluate the effectiveness and efficiency of DiaVio, we design the following five research questions.

- **RQ1**: How effective is DiaVio in liability determination on real-world accident reports?
- **RQ2**: How effective is DiaVio in crash classification on real-world accident reports?
- **RQ3**: How effective and efficient is DiaVio in diagnosing violation scenarios, compared with manual diagnosis?
- **RQ4**: What are the types of violations DiaVio helps to find?
- **RQ5**: What is the quality of accident report parsing and the consistency of our LLM-empowered violation diagnosis?

3.1 Evaluation Setup

Datasets. As introduced in Sec. 2.2, we use two datasets of accident reports. The NMVCCS dataset [5] includes 6,949 cases of vehicle accidents, composed of crash summary, causation and category analysis. The CIREN dataset [3] includes 2,499 cases of vehicle accidents composed of crash summary and causation. Due to web page failures, we successfully obtain 6,947 NMVCCS cases and 2,490 CIREN cases. Excluding the cases that do not reveal which vehicle is responsible, we obtain 6,910 NMVCCS cases and 2,455 CIREN cases.

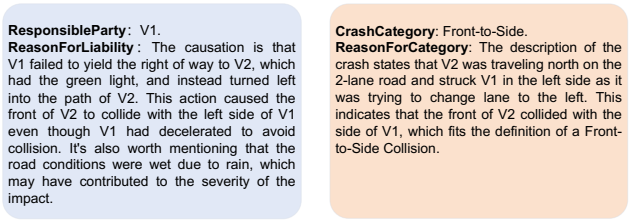
Base LLMs. We use seven open-source LLMs that belong to four different series. **Baichuan2-7B-Chat** [57] is a part of Baichuan 2, a series of large-scale multilingual language models developed by Baichuan Intelligent Technology. It contains 7 billion parameters and is trained from scratch on a massive corpus of 2.6 trillion tokens. **ChatGLM3-6B** [7], released by Zhipu.AI and Tsinghua University's KEG lab, is a part of the ChatGLM3 series, which is the latest generation of open-source, bilingual chat-oriented LLMs. **Qwen-7B/14B-Chat** [16, 17], designed by Alibaba Cloud, are parts of the Qwen series. These models are trained on an extensive dataset comprising web texts, literary works and code. **Llama-2-7b/13b/70b-chat-hf** [46–48] are released by Meta AI. The Llama2 series is a collection of pre-trained and fine-tuned generative text models ranging in scale from 7 billion to 70 billion parameters.

Evaluation Metrics. We use three metrics, i.e., **accuracy**, **precision** and **recall**, to assess the diagnosis effectiveness (with respect to the *ResponsibleParty* and *CrashCategory* in our DSL). In addition, we use several lexical and semantic metrics to assess the diagnosis explanation effectiveness (with respect to the *ReasonForLiability* and *ReasonForCategory* in our DSL). In other words, we compare the generated explanation text with the ground truth explanation text from both lexical and semantic perspectives.

For lexical metrics, we employ **BLEU-4** (Bilingual Evaluation Understudy) [39] to calculate n-grams overlap from 1 to 4 between the ground truth and generated explanation texts. We also use **METEOR** (Metric for Evaluation of Translation with Explicit Ordering) [38], which is based on the harmonic mean of unigram precision and recall as well as stemming and synonymy matching, and **ROUGE-L** (Recall Oriented Understudy for Gisting Evaluation) [10], which considers sentence-level structural similarity.

For semantic metrics, we use **BERTScore** [61], which utilizes the pre-trained contextual embeddings from BERT [18] and matches ground truth and generated explanation texts based on cosine similarity. We also employ **BLEURT** [55] to measure the degree to what extent the generated explanation text is fluent and conveys the meaning of the ground truth explanation. We further use **NUBIA** (NeUral Based Interchangeability Assessor) [31], which incorporates the semantic similarity, logical inference and sentence legibility from exposing layers of pre-trained language models.

Research Question Setup. For **RQ1**, we use both NMVCCS and CIREN datasets, a total of 9,365 cases of accidents. We use 6,275, 1,545 and 1,545 cases for training, validation and testing. For **RQ2**, we only use the NMVCCS dataset because the CIREN dataset does not provide the crash category analysis. We divide each category of accidents by a ratio of 4:1:1, and obtain 4,633, 1,137 and 1,140 cases for training, validation and testing. For **RQ1** and **RQ2**, we initially assess the effectiveness of the seven base LLMs on our diagnosis tasks.



(a) Liability Determination (b) Crash Classification

Figure 4: Diagnosis of Fine-Tuned Llama2-70b-chat-hf

Subsequently, we select three top-performing base LLMs for fine-tuning with our dataset, ultimately obtaining the best-performing model on the testing data. We test each model on testing data for two times and report the average results.

For **RQ3** and **RQ4**, we bridge DiaVio with two state-of-the-art open source scenario-based testing approaches, i.e., AV-Fuzzer [36] and DriveFuzz [34]. Specifically, we set up AV-Fuzzer with the Apollo 8.0 ADS [9] and the SORA-SVL simulator [26], and set up DriveFuzz with the Behavior Agent ADS [13] and the Carla simulator [20] according to their documents respectively. Our experiments are not designed to find more violations, but to evaluate whether DiaVio can effectively and efficiently diagnose violations and whether the violations are caused by ADSs. Therefore, we run AV-Fuzzer and DriveFuzz for 24 hours. Then, two of the authors separately diagnose each violation to establish the ground truth. A third author is involved for a group discussion to resolve conflicts and reach agreements. Finally, the Cohen Kappa coefficient reaches 0.897.

For **RQ5**, to measure the quality of accident report parsing, two of the authors independently assess the accuracy of each key-value pair responded by GPT-4 given the original accident reports. When there is a conflict between the two authors' assessment, a third author is involved to have a group discussion for reaching agreements. Finally, the Cohen Kappa coefficient reaches 0.845. Besides, to measure the consistency of our LLM-empowered violation diagnosis, we compare the diagnostic results of two inferences on each testing data in **RQ1** and **RQ2** and on each detected violation in **RQ3**.

Environment. We conduct model fine-tuning and inference on a Ubuntu 20.04.6 LTS server with 8 NVIDIA A800 GPUs, Intel Core 8358P CPU with 2.60GHz processor and 1TB memory. We run AV-Fuzzer and DriveFuzz on a Ubuntu 18.04.6 LTS server with a NVIDIA GeForce RTX 3090 GPU, Intel Core i9-12900K CPU with 5.20GHz processor and 64GB memory.

3.2 Results of RQ1

We measure the effectiveness of both base and fine-tuned LLMs on the task of liability determination, i.e., determining which vehicle is responsible for the accident as well as explaining the reason, using the 1,545 accidents in the testing data. Fig. 4(a) illustrates an example of the diagnostic result inferred by a fine-tuned LLM.

Effectiveness of Base LLMs. Table 2(a) reports the effectiveness of our seven base LLMs. In terms of accuracy, it measures the proportion of accidents whose inferred *ResponsibleParty* is the same to the ground truth from the total number of accidents in the testing data. The Qwen series performs poorly, having an accuracy of only about 50%, and the Llama2 series performs well, achieving an accuracy of

Table 2: Effectiveness on the Liability Determination Task

(a) **Results of Base LLMs**

Models	Accuracy	Lexical Metrics			Semantic Metrics		
		BLUE-4	METEOR	ROUGE-L	BERTScore	BLEURT	NUBIA
Baichuan2-7B-Chat	61.68	0.09	8.35	9.16	54.47	39.56	32.99
ChatGLM3-6B	67.44	0.81	11.02	16.29	61.86	38.92	32.36
Qwen-7B-Chat	52.56	2.08	14.53	16.02	63.52	43.69	36.01
Qwen-14B-Chat	47.38	1.20	12.43	15.43	62.95	43.43	38.39
Llama2-7b-chat-hf	78.25	2.69	13.04	18.96	63.73	41.76	37.59
Llama2-13b-chat-hf	70.74	1.82	13.16	17.38	63.93	42.42	38.16
Llama2-70b-chat-hf	81.29	1.71	15.20	19.45	64.24	45.46	44.17

(b) **Results of Fine-Tuned LLMs**

Models	Accuracy	Lexical Metrics			Semantic Metrics		
		BLUE-4	METEOR	ROUGE-L	BERTScore	BLEURT	NUBIA
Llama-2-7b-chat-hf	82.91	4.66	17.61	20.58	72.79	44.71	55.05
Llama-2-13b-chat-hf	82.52	5.57	18.68	22.00	72.88	45.15	57.47
Llama-2-70b-chat-hf	87.06	6.96	19.28	23.26	74.34	45.53	58.54

over 70%, with Llama2-70B-chat-hf achieving the highest accuracy of 81.29%. In terms of lexical and semantic metrics on the inferred *ReasonForLiability*, they are measured on the accidents whose inferred *ResponsibleParty* is the same to the ground truth. Baichuan2-7B-Chat performs the worst, with its metrics significantly lower than other models. Llama2-7b-chat-hf performs the best on BLUE-4, achieving 2.69, while Llama2-70b-chat-hf performs the best on METEOR, ROUGE-L, BERTScore, BLEURT and NUBIA, scoring 15.20, 19.45, 64.24, 45.46 and 44.17, respectively. Overall, the base LLMs in the Llama2 series performs well on the liability determination task.

We observe that the performance of Qwen-14B-Chat and Llama2-13b-chat-hf do not significantly improve with an increase in their model parameters. A contributing factor to this phenomenon appears to be the models' tendency to provide more conservative responses in certain cases, such as stating "hard to judge due to legal, ethical, or moral aspects". Additionally, these models sometimes offer lengthy explanations. Such styles of responses could be attributed to the models' pre-training process, where they are trained to consider legal and ethical factors. As a result, it leads to a decrease in their performance when processing certain cases.

Besides, upon analyzing the performance differences across models in the task of liability determination, we identified several critical factors that impacted model's effectiveness. Specifically, models such as Baichuan2-7B-Chat and the Qwen series exhibited a lower accuracy, which can be attributed to their insufficient comprehension of traffic scenarios and task demands. For example, Baichuan2-7B-Chat's outputs were often ambiguous and lacked the clarity and precision needed for assigning responsibility in complex traffic situations. The Qwen series demonstrated a tendency to provide indecisive or overly cautious responses, often suggesting a lack of sufficient information to make a direct judgment or attributing responsibility to all parties involved without pinpointing the primary responsible party. This behavior could be traced back to the models' training process, where they might have been exposed to a balanced approach towards legal, ethical, or moral considerations, leading them to shy away from definitive statements in complex or ambiguous situations. It is also worth mentioning that accuracy and linguistic metrics (such as BLEU-4, METEOR, and ROUGE-L) can be not direct. Models demonstrating a lower accuracy in liability determination may still produce outputs that are structurally and linguistically similar to the ground truth answers.

Effectiveness of Fine-Tuned LLMs. We fine-tune the three top-performing base LLMs. After fine-tuning, these models achieve a significant improvement in all the metrics as shown in Table 2(b). In terms of accuracy, these models achieve an average increase of 7.40%, obtaining an accuracy of up to 87.06%. In terms of lexical metrics, these models achieve an average increase of 3.66%, 4.72% and 3.35% on BLUE-4, METEOR and ROUGE-L, respectively. In terms of semantic metrics, these models achieve an average increase of 9.37%, 1.92% and 17.05% on BERTScore, BLEURT and NUBIA, respectively.

Summary. Fine-tuning enables base LLMs to gain a better capability of liability determination by not only accurately determining the responsible party but also explaining the reason. The fine-tuned Llama2-70b-chat-hf performs the best in all the metrics, and is effective in the liability determination task.

3.3 Results of RQ2

We measure the effectiveness of both base and fine-tuned LLMs on the task of crash classification, i.e., classifying each accident into a specific category as well as explaining the reason, using the 1,140 accidents in the NMVCCS testing data. Fig. 4(b) illustrates an example of the diagnostic result inferred by a fine-tuned LLM.

Effectiveness of Base LLMs. Table 3(a) shows the effectiveness of our seven base LLMs. In terms of accuracy, it measures the proportion of accidents whose inferred *CrashCategory* is the same to the ground truth from the total number of accidents in the testing data. Baichuan2-7B-Chat performs the worst, hardly classifying accidents correctly, with an accuracy of only 17.41%. Llama-2-70b-chat-hf performs the best with an accuracy of 76.17%. The accuracy of other models falls between 30% and 60%. It is potentially

Table 3: Effectiveness on the Crash Classification Task

(a) **Results of Base LLMs**

Models	Accuracy	Lexical Metrics			Semantic Metrics		
		BLUE-4	METEOR	ROUGE-L	BERTScore	BLEURT	NUBIA
Baichuan2-7B-Chat	17.41	2.55	13.30	15.05	48.22	21.92	14.24
ChatGLM3-6B	30.43	1.09	11.87	13.34	47.93	26.46	12.43
Qwen-7B-Chat	29.20	3.97	17.78	21.38	48.56	44.40	31.20
Qwen-14B-Chat	37.90	2.56	15.55	18.32	48.06	41.60	30.96
Llama-2-7b-chat-hf	39.14	7.22	22.34	23.93	50.82	47.45	40.43
Llama-2-13b-chat-hf	53.56	7.13	24.10	24.43	50.97	48.08	40.80
Llama-2-70b-chat-hf	76.17	7.68	25.51	25.25	52.02	49.29	41.62

(b) **Results of Fine-Tuned LLMs**

Models	Accuracy	Lexical Metrics			Semantic Metrics		
		BLUE-4	METEOR	ROUGE-L	BERTScore	BLEURT	NUBIA
Llama-2-7b-chat-hf	67.19	17.76	30.09	38.87	65.70	55.49	42.56
Llama-2-13b-chat-hf	77.04	15.13	30.42	35.02	65.22	55.83	43.89
Llama-2-70b-chat-hf	85.44	18.22	29.93	38.71	65.33	56.45	45.22

because base LLMs fail to understand the six crash categories in the testing data, resulting in a lower accuracy. In terms of lexical and semantic metrics on the inferred *ReasonForCategory*, they are measured on the accidents whose inferred *CrashCategory* is the same to the ground truth. ChatGLM3-6B performs the worst. The Llama2 series outperforms other models on these metrics. The reasons provided by Llama2 models are closer to the ground truth. Among them, Llama-2-70b-chat-hf is the best base model, achieving 7.68, 25.51, 25.25, 52.02, 49.29 and 41.72 on BLUE-4, METEOR, ROUGE-L, BERTScore, BLEURT and NUBIA, respectively.

We observed a common misclassification between Front-to-Side Collisions and Rear-End Collisions by most base LLMs except for Llama-2-13b/70b-chat-hf. Despite the clear definitions being provided for each crash category, base LLMs still misinterpreted these two types of scenarios occurring at intersections. Moreover, base LLMs may classify violations whose types they could not determine as "Other". These inaccuracies suggest that, without fine-tuning, base LLMs may struggle to differentiate between violation types that share similar elements but occur under different circumstances.

Effectiveness of Fine-Tuned LLMs. Table 3(b) shows the results of fine-tuned Llama2 models on crash classification. After fine-tuning, these three models show a significant improvement in all metrics. In terms of accuracy, these models achieve an average improvement of 20.27%, with the highest reaching 85.44%. In terms of lexical metrics, these models achieve average an improvement of 9.69%, 6.16% and 13.00% on BLUE-4, METEOR and ROUGE-L, respectively. In terms of semantic metrics, these models show an average improvement of 14.15%, 7.65% and 2.94% on BERTScore, BLEURT and NUBIA, respectively.

Summary. Fine-tuning enables base LLMs to have a deeper understanding of crash classification by not only accurately classifying the crash category but also explaining the reason. Llama2-70b-chat-hf performs the best in all the metrics, and is effective in the crash classification task.

Table 4: Results of DiaVio Compared with Manual Diagnosis on the Liability Determination Task

Responsible Party	AV-Fuzzer		DriveFuzz	
	DiaVio	Manual	DiaVio	Manual
EGO Vehicle	54	41	9	9
NPC Vehicle	138	151	10	10

3.4 Results of RQ3

In 24 hours of simulation testing, AV-Fuzzer generates 192 violation scenarios, while DriveFuzz generates 19 violation scenarios. Two of the authors take three days to manually diagnose these violations to establish the ground truth, while DiaVio, with the best-performing fine-tuned Llama2-70b-chat-hf takes two hours to automatically diagnose these violations. Therefore, DiaVio can significantly reduce the manual effort of diagnosing violation scenarios.

Table 4 reports comparison results of DiaVio and manual diagnosis on the liability determination task. For AV-Fuzzer, DiaVio determines that 54 violations are caused by the EGO vehicle (i.e., the ADS), and 138 violations are caused by NPC vehicles. Instead, our manual diagnosis suggests that 41 violations are caused by the EGO vehicle, and 151 are caused by NPC vehicles. In summary, DiaVio wrongly attributes 13 violations caused by NPC vehicles to the responsibility of the EGO vehicle. In these 13 scenarios, NPC vehicles overtake and change lanes in front of the EGO vehicle, nearly completing the maneuver (with a close distance between the vehicles). The EGO vehicle fails to brake in time, leading to a collision. Our manual diagnosis thinks it is due to NPC vehicle's unreasonable lane changing and overtaking. However, DiaVio considers them to be a rear-end collision caused by the EGO vehicle. For DriveFuzz, DiaVio determines that 9 violations are caused by the EGO vehicle and 10 violations are caused by the NPC vehicles, which is totally consistent with manual diagnosis. Overall, DiaVio achieves an accuracy of 93.84%, a precision of 79.36% and a recall of 100% on liability determination task.

Table 5: Results of DiaVio Compared with Manual Diagnosis on the Crash Classification Task

Crash Category	AV-Fuzzer		DriveFuzz	
	DiaVio	Manual	DiaVio	Manual
Rear-End Collision	47	42	3	5
Frontal Collision	0	0	3	3
Front-to Side Collision	144	150	13	11
Others	1	0	0	0

Table 6: Effectiveness of DiaVio w.r.t. Crash Categories

Crash Category	AV-Fuzzer			DriveFuzz		
	Pre.	Rec.	Acc.	Pre.	Rec.	Acc.
Rear-End Collision	89.36%	100%	96.88%	100%	60.00%	89.47%
Frontal Collision	N/A	N/A		100%	100%	
Front-to-Side Collision	100%	96.00%		84.62%	100%	
Others	0.00%	N/A		N/A	N/A	

Table 5 reports comparison results of DiaVio and manual diagnosis on the crash classification task. For AV-Fuzzer, DiaVio incorrectly classifies 5 front-to-side collisions to rear-end collisions. In these 5 scenarios, the rear vehicle hits the front vehicle that almost finishes lane changing, causing DiaVio to mistakenly classify them as rear-end collisions. DiaVio also fails to recognize one front-to-side collision, and thus incorrectly classifies it as others. For DriveFuzz, DiaVio incorrectly classifies 2 rear-end collisions as front-to-side collisions. In these 2 scenarios, the rear vehicle collides with the side-rear of the front vehicle that is trying to turn, causing DiaVio to mistakenly classify them as front-to-side collisions. Table 6 shows the accuracy, precision and recall of DiaVio with respect to crash categories. Overall, it has an accuracy of 96.21%, a macro-precision of 72.18% and a macro-recall of 97.34% on crash classification task.

We also ask two of the authors who carry out the manual diagnosis to review the reasons provided by DiaVio. Although DiaVio may not provide perfect reasons and even sometimes lead to a wrong direction, they still think these reasons can help them quickly diagnose violations, serving a guiding role.

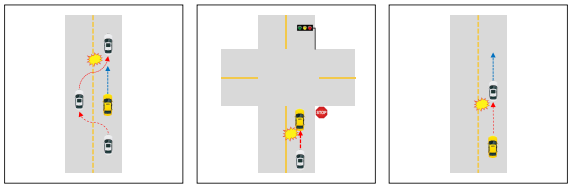
Summary. DiaVio achieves an accuracy of 93.84% and 96.21% on the liability determination and crash classification tasks efficiently. Therefore, DiaVio can eliminate a large number of violation scenarios caused by NPC vehicles, significantly improving the efficiency of violation diagnosis.

3.5 Results of RQ4

Based on the diagnostic results of liability determination and crash classification, we summarize common types of violations caused by NPC vehicles and the EGO vehicle for each crash category. To illustrate each case, we use a yellow car to represent an EGO vehicle and a white car to represent a NPC vehicle.

In rear-end collisions, we summarize three main types of collision scenarios, two of which are caused by NPC vehicles, and one of which are caused by the EGO vehicle due to flaws in the ADS.

Case Study 1: Fig. 5(a) shows one type of collisions caused by NPC vehicles in rear-end collisions. NPC vehicle performs a risky overtaking and lane-changing maneuver. EGO vehicle maintains its lane and decelerates in response to the erratic behavior of NPC vehicle. Despite EGO vehicle's attempt to brake, NPC vehicle cuts in too closely, leading to a collision where EGO vehicle rear-ends NPC vehicle. In this situation, EGO vehicle recognizes the lane-changing and overtaking behavior of NPC vehicle, and brakes to avoid it. However, NPC vehicle fails to maintain sufficient distance from EGO vehicle after overtaking, leading to a rear-end collision.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 5: Rear-End Crashes

Case Study 2: As depicted in Fig. 5(b), EGO vehicle arrives at an intersection, recognizes the traffic signal as red, decelerates, and stops in front of the stop line to wait. NPC vehicle, maintaining its original speed, rear-ends EGO vehicle from behind. In this situation, it is evident that the collision is caused by NPC vehicle.

Case Study 3: As shown in Fig. 5(c), EGO vehicle fails to take actions to decelerate and brake, instead colliding with NPC vehicle at a constant speed, when the leading NPC vehicle slows down to a stop. In this situation, EGO vehicle fails to maintain a safe distance from NPC vehicle ahead, and the ADS does not recognize NPC vehicle's deceleration, leading to a rear-end collision.

In front-to-side collisions, we summarize six main types of collision scenarios, four of which are caused by NPC vehicles, and two of which are caused by the EGO vehicle due to flaws in the ADS.

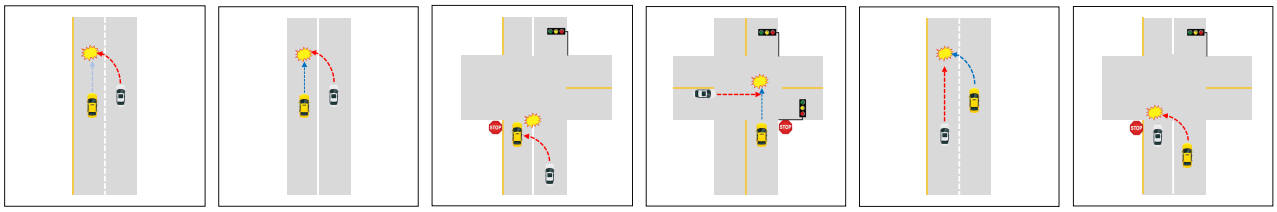
Case Study 4: As depicted in Fig. 6(a), NPC vehicle performs an abrupt lane changing, which leads to a sideswipe collision with EGO vehicle. In this situation, although EGO vehicle takes actions to decelerate and evade, the collision cannot be avoided due to NPC vehicle's sudden lane-changing behavior. Such accidents are entirely the responsibility of NPC vehicle.

Case Study 5: Fig. 6(b) shows a scenario where NPC vehicle violates traffic regulations by crossing a solid line to change lanes, colliding with EGO vehicle running in an adjacent lane. In this scenario, it is clearly caused by the irrational behavior of NPC vehicle.

Case Study 6: Fig. 6(c) illustrates an accident where EGO vehicle stops and is waiting for a red light at the intersection. However, NPC vehicle takes a lane changing and consequently impacts the side of EGO vehicle. In this scenario, it is clearly caused by the irrational behavior of NPC vehicle.

Case Study 7: As shown in Fig. 6(d), EGO vehicle initially stops at a red light at an intersection. When the light turns green, EGO vehicle assumes that NPC vehicle would obey the red light and stop for waiting, hence it accelerates through the intersection. However, NPC vehicle does not obey the red light, resulting in a side collision.

Case Study 8: As shown in Fig. 6(e), EGO vehicle attempts to change lanes to the left, but fails to detect an approaching NPC vehicle from behind, leading to a side collision with NPC vehicle. In this situation, the ADS fails to successfully predict the behavior of vehicles in the adjacent lane and makes a lane change at a too short distance, attributing the collision to EGO vehicle.



(a) Case 4 (b) Case 5 (c) Case 6 (d) Case 7 (e) Case 8 (f) Case 9

Figure 6: Front-to-Side Crashes

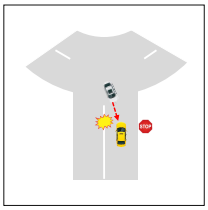


Figure 7: Frontal Crashes (Case 10)

Case Study 9: As depicted in Fig. 6(f), EGO vehicle makes a left turn at an intersection, but collides with NPC vehicle, which stops slightly beyond the stop line. In this situation, the ADS fails to detect the vehicle stopped at the intersection, leading to a collision.

In frontal collisions, they are all caused by NPC vehicles, and we summarize one type of collision scenario.

Case Study 10: As shown in Fig. 7, EGO vehicle slows down and stops at the entrance of a roundabout, waiting to enter. NPC vehicle exits the roundabout and collides head-on with EGO vehicle.

Summary. We summarize common types of scenarios that are caused by EGO and NPC vehicles with the help of our diagnostic results. In practice, liability determination helps to eliminate violations caused by NPC vehicles, and crash classification helps to quickly summarize common types without the need to look into each violation of the same crash category.

3.6 Results of RQ5

Quality of Accident Report Parsing. Due to the large number of accident reports, we evaluate the quality of GPT-4's parsing (see Sec. 2.2) through a sampling approach. We randomly select 369 accident reports from a total of 9,365 accident reports, achieving a confidence level of 95% and a margin error of 5%. Here, we use accuracy, which is defined as the proportion of the accident reports where GPT-4's parsing output matches the actual information from accident reports, to measure the quality. We compute the accuracy for each of the five components, as reported in Table 7.

We can see that GPT-4 achieves an accuracy of over 85% for all the five components, and achieves an accuracy of over 95% for ***Road***, ***Environment*** and ***Diagnosis***. Overall, the average accuracy is 93.41%. Specifically, ***Diagnosis*** exhibits the highest accuracy of 98.17%, attributed to the fact that traffic accident reports typically state the responsible party and crash analysis clearly. ***Vehicles*** exhibits the lowest accuracy of 85.36%. This is partially because some reports provide implicit details on vehicle positions and behaviors. In such instances, GPT-4 often outputs "unmentioned" as it fails to understand the implicit details, leading to a lower accuracy in ***Vehicles***.

Table 7: Quality of GPT-4's Accident Report Parsing

	Road	Environment	Obstacles	Vehicles	Diagnosis
Accuracy	96.34%	97.56%	89.63%	85.36%	98.17%

Table 8: Lexical and Semantic Comparison of the Reasons Generated by Fine-Tuned Llama2-70b-chat-hf

BLUE-4	METEOR	ROUGE-L	BERTScore	BLEURT	NUBIA
28.65	31.68	45.35	97.69	59.62	59.97

Consistency of Diagnosis. We compare the results of two inferences by fine-tuned Llama2-70b-chat-hf on the same data. Here we report the consistency results on the testing data in **RQ1** and **RQ2**, and the results on the detected violations in **RQ3** are similar and thus are omitted. With respect to *ResponsibleParty* and *CrashCategory*, the model's consistency reaches 96.88% and 89.47%, respectively. With respect to *ReasonForLiability* and *ReasonForCategory*, we measure the lexical and semantic similarity between the outputs of two inferences, as reported in Table 8. We can observe that the two inferences have a high similarity in both lexical and semantic metrics.

Summary. GPT-4 accurately parse the crash information from traffic accident reports, which helps to enhance the performance of base LLMs on our diagnosis task through fine-tuning. Besides, DiaVio, empowered by LLM, can produce consistent diagnostic results.

3.7 Threats to Validity

First, the quality of real-world accident reports and the converted dataset using GPT-4 pose a threat to validity. To that end, we select the NMVCCS and CIREN datasets, and evaluate the quality of the converted data through manual analysis. We believe our approach can be applied to other traffic accident datasets that include detailed descriptions of accident scenes and police analysis results.

Second, the performance discrepancies of different LLMs on our task affect the validity. Thus, we evaluate seven different LLMs with varying parameter sizes, and fine-tune the three best-performing LLMs. Ultimately, we use the best-performing Llama2-70b-chat-hf model. Users can choose models with smaller parameter sizes based on their environmental conditions, without sacrificing too much performance. All the fine-tuned LLMs are open-sourced and available at our replication site.

Third, the general applicability of DiaVio poses another threat. To alleviate this threat, we bridge DiaVio with AV-Fuzzer and DriveFuzz, demonstrating the effectiveness of DiaVio across two

simulation testing approaches, two simulators, and two ADSs. We believe that DiaVio can be easily adapted to support other simulation testing approaches.

Last, we summarize our DSL from real-world accident reports. Our DSL is designed to encompass a broad range of components and subcomponents based on accident reports in NMVCCS and CIREN dataset. When preparing the ground truth for **RQ5**, we manually analyze the 369 accident reports, and our DSL can fully describe these accidents. Therefore, we believe our DSL is reasonable. The content of the subcomponents can be easily extended when more information is present in other dataset.

4 RELATED WORK

Scenario Description Language. DSLs have been proposed to generate driving scenarios for testing ADSs. For example, Fremont et al. [21] introduce Scenic as a DSL to characterize driving scenarios based on a probabilistic programming method. Queiroz et al. [53] design GeoScenario as a DSL for scenario representation to substantiate testing scenarios in simulation. OpenScenario [6] is the current XML-based standard for scenario description. These existing DSLs mostly aim at generating testing scenarios and thus contain complicated and hard-to-master syntax. In contrast to existing DSLs, we propose the crash DSL with simple syntax, specifically designed for describing crashes in accident reports and violation scenarios.

Scenario-Based Testing. Scenario-based testing [19, 23, 37, 41, 62, 65] has been widely studied to generate diverse driving scenarios for ADS testing. Several attempts have been made to find safety violations that cause crashes. For example, Abdessalem et al. [1, 11], Li et al. [36], and Tian et al. [58] define a driving scenario as a vector of multiple variables (e.g., vehicle speed) and apply generic algorithms to search the variable space for driving scenarios where the EGO vehicle can collide with NPC vehicles. Chen et al. [14] design an adaptive evaluation framework to find crashes in adversarial environments generated by deep reinforcement learning, and they specifically focus on lane-changing scenarios. Zhong et al. [64] propose AutoFuzz to use neural networks in the process of evolutionary search to generate more complex and valid collision scenarios. Tian et al. [59] propose to generate safety-critical driving scenarios by mining influential behavior patterns from real traffic trajectories. Lu et al. [42, 43] use reinforcement learning to learn environment configurations that lead to a crash. Hildebrandt et al. [24] introduce a physical environment-state coverage metric to find crashes.

In addition to focusing on crashes, some studies further aim to identify driving scenarios where ADSs violate established rules. For example, Abdessalem et al. [2] use a search-based method to generate scenarios where vehicles exhibit behaviors like speeding, unsafe lane changes, fast acceleration, and hard braking. Gladisch et al. [22] apply search-based testing to three ADS settings, adaptive cruise control, lane keeping and steering control. Luo et al. [44] use an evolutionary many-objective optimization algorithm to generate driving scenarios that expose as many requirement violation patterns as possible. Kim et al. [34] propose DriveFuzz to mutate actors' actions, road condition and weather condition, aiming to find collisions and safety-critical traffic violations (e.g., running red lights). Sun et al. [56], Zhang et al. [63] and Li et al. [35] propose to generate driving scenarios that break specific traffic rules. Huai et al. [27] aim to generate valid and effective driving scenarios that cause comfort and safety violations. Cheng et al. [15] propose to explore the diversity of vehicle behavior to detect diverse violations. Although these studies focus on different violations, the exploration of crash scenarios has always been an indispensable part.

To the best of our knowledge, all these previous approaches do not distinguish whether crashes are caused by ADSs or not, making the generated violation scenarios less useful for ADS developers. Our work aims to equip these previous approaches with the diagnosis capability so that ADS developers can efficiently focus on more useful violation scenarios. Huai et al.'s work [28] is the closest work to ours. They first manually diagnose generated violation scenarios, and surprisingly find that none of them is caused by ADSs. To only find violation scenarios that are caused by ADSs, they opt to bridge multiple ADSs for interaction rather than using NPC vehicles. However, this is achieved at the cost of feeding ground truth directly into the ADSs' localization and perception modules and only testing the planning module. Differently, our work is a general approach.

5 CONCLUSIONS

We have proposed and implemented DiaVio to automatically diagnose safety violations in simulation testing by leveraging large language models (LLMs) and a new domain specific language (DSL) of crash. Large-scale experiments have been conducted to demonstrate the effectiveness and efficiency of DiaVio. In future, we plan to extend DiaVio to support more simulation testing approaches and establish a framework to assess them. Moreover, we also plan to support more types of violations apart from crashes.

6 DATA AVAILABILITY

All the experimental data and source code of our work is available at our replication site https://diavio.github.io/diavio/.

ACKNOWLEDGMENTS

This work was supported by the National Science and Technology Major Project (2021ZD0112903).

REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-Based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering*. 1016–1026.

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 143–154.

[3] National Highway Traffic Safety Administration. 2016. *CIREN (2004-2015) search.* Retrieved November 15, 2023 from https://crashviewer.nhtsa.dot.gov/LegacyCIREN/Search

[4] National Highway Traffic Safety Administration. 2016. *NHTSA Crash Viewer.* Retrieved November 15, 2023 from https://www.nhtsa.gov/

[5] National Highway Traffic Safety Administration. 2016. *NMVCCS (2005-2007) search.* Retrieved November 15, 2023 from https://crashviewer.nhtsa.dot.gov/LegacyNMVCCS/Search

[6] ASAM. 2021. *ASAM OpenSCENARIO: User Guide.* Retrieved November 13, 2023 from https://www.asam.net/index.php?eID=dumpFile&t=f&f=4092&token=d3b6a55e911b22179e3c0895fe2caae8f5492467

[7] Knowledge Engineering Group (KEG) & Data Mining at Tsinghua University. 2023. *ChatGLM3-6B.* Retrieved November 26, 2023 from https://huggingface.co/THUDM/chatglm2-6b

[8] Daniel Atherton. 2022. Incident 434: Sudden Braking by Tesla Allegedly on Self-Driving Mode Caused Multi-Car Pileup in Tunnel. In *AI Incident Database*,

Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/434/

[9] Baidu. 2022. *Apollo: An open autonomous driving platform.* Retrieved November 15, 2023 from https://github.com/ApolloAuto/apollo

[10] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. 65–72.

[11] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2016. Testing Advanced Driver Assistance Systems Using Multi-Objective Search and Neural Networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 63–74.

[12] Assaf Botzer, Oren Musicant, and Yaniv Mama. 2019. Relationship between hazard-perception-test scores and proportion of hard-braking events during on-road driving–An investigation using a range of thresholds for hard-braking. *Accident Analysis & Prevention* 132 (2019), 105267.

[13] CARLA. 2023. *CARLA Agents.* Retrieved December 15, 2023 from https://carla.readthedocs.io/en/0.9.12/adv_agents/

[14] Baiming Chen, Xiang Chen, Qiong Wu, and Liang Li. 2021. Adversarial evaluation of autonomous vehicles in lane-change scenarios. *IEEE Transactions on Intelligent Transportation Systems* 23, 8 (2021), 10333–10342.

[15] 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.

[16] Alibaba Cloud. 2023. *Qwen-14B-Chat.* Retrieved November 26, 2023 from https://huggingface.co/Qwen/Qwen-14B-Chat

[17] Alibaba Cloud. 2023. *Qwen-7B-Chat.* Retrieved November 26, 2023 from https://huggingface.co/Qwen/Qwen-7B-Chat

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 4171–4186.

[19] Wenhao Ding, Chejian Xu, Mansur Arief, Haohong Lin, Bo Li, and Ding Zhao. 2023. A Survey on Safety-Critical Driving Scenario Generation—A Methodological Perspective. *IEEE Transactions on Intelligent Transportation Systems* 24, 7 (2023), 6971–6988.

[20] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

[21] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2019. Scenic: A Language for Scenario Specification and Scene Generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 63–78.

[22] Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. 2020. Experience Paper: Search-Based Testing in Automated Driving Control Applications. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*. 26–37.

[23] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. 2020. Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study. In *Proceedings of the IEEE 13th International Conference on Software Testing, Validation and Verification*. 85–95.

[24] Carl Hildebrandt, Meriel von Stein, and Sebastian Elbaum. 2023. PhysCov: Physical Test Coverage for Autonomous Vehicles. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 449–461.

[25] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *Proceedings of the 10th International Conference on Learning Representations*.

[26] Yuqi Huai. 2023. *SORA-SVL.* Retrieved September 13, 2023 from https://www.ics.uci.edu/~yhuai/SORA-SVL/

[27] Y. Huai, S. Almanee, Y. Chen, X. Wu, Q. Chen, and J. Garcia. 2023. scenoRITA: Generating Diverse, Fully Mutable, Test Scenarios for Autonomous Vehicle Planning. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4656–4676.

[28] 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 *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering*. 2591–2603.

[29] Pengliang Ji, Ruan Li, Yunzhi Xue, Qian Dong, Limin Xiao, and Rui Xue. 2021. Perspective, survey and trends: Public driving datasets and toolsets for autonomous driving virtual test. In *Proceedings of the IEEE International Intelligent Transportation Systems Conference*. 264–269.

[30] Nidhi Kalra and Susan M Paddock. 2016. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice* 94 (2016), 182–193.

[31] Hassan Kane, Muhammed Yusuf Kocyigit, Ali Abdalla, Pelkins Ajanoh, and Mohamed Coulibali. 2020. NUBIA: NeUral based interchangeability assessor for text generation. In *Proceedings of the 1st Workshop on Evaluating NLG Evaluation*. 28–37.

[32] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. 2021. A survey on simulators for testing self-driving cars. In *Proceedings of the 4th International Conference on Connected and Autonomous Driving*. 62–70.

[33] Srishti Khemka. 2021. Incident 347: Waymo Self-Driving Taxi Behaved Unexpectedly, Driving away from Support Crew. In *AI Incident Database*, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/347/

[34] Seulbae Kim, Major Liu, Junghwan" John" Rhee, Yuseok Jeon, Yonghwi Kwon, and Chung Hwan Kim. 2022. Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1753–1767.

[35] Changwen Li, Joseph Sifakis, Qiang Wang, Rongjie Yan, and Jian Zhang. 2023. Simulation-Based Validation for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 842–853.

[36] 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 *Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering*. 25–36.

[37] Li Li, Wu-Ling Huang, Yuehu Liu, Nan-Ning Zheng, and Fei-Yue Wang. 2016. Intelligence testing for autonomous vehicles: A new approach. *IEEE Transactions on Intelligent Vehicles* 1, 2 (2016), 158–166.

[38] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the Workshop on Text Summarization Branches Out*. 74–81.

[39] Chin-Yew Lin and Franz Josef Och. 2004. Orange: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of the 20th International Conference on Computational Linguistics*. 501–507.

[40] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of the 7th International Conference on Learning Representations*.

[41] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 31–43.

[42] Chengjie Lu. 2023. Test Scenario Generation for Autonomous Driving Systems with Reinforcement Learning. In *Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings*. 317–319.

[43] Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue, and Shaukat Ali. 2023. Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions. *IEEE Transactions on Software Engineering* 49, 1 (2023), 384–402.

[44] Yixing Luo, Xiao-Yi Zhang, Paolo Arcaini, Zhi Jin, Haiyan Zhao, Fuyuki Ishikawa, Rongxin Wu, and Tao Xie. 2021. Targeting requirements violations of autonomous driving systems by dynamic evolutionary search. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering*. 279–291.

[45] Sean McGregor. 2018. Incident 4: Uber AV Killed Pedestrian in Arizona. In *AI Incident Database*, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February 13, 2023 from https://incidentdatabase.ai/cite/4

[46] Meta. 2023. *Llama-2-13b-chat-hf.* Retrieved November 26, 2023 from https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

[47] Meta. 2023. *Llama-2-70b-chat-hf.* Retrieved November 26, 2023 from https://huggingface.co/meta-llama/Llama-2-70b-chat-hf

[48] Meta. 2023. *Llama-2-7b-chat-hf.* Retrieved November 26, 2023 from https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

[49] Wassim G Najm, Raja Ranganathan, Gowrishankar Srinivasan, John D Smith, Samuel Toma, Elizabeth D Swanson, August Burgett, et al. 2013. *Description of light-vehicle pre-crash scenarios for safety applications based on vehicle-to-vehicle communications.* Technical Report. United States. Department of Transportation. National Highway Traffic Safety.

[50] Wassim G Najm, John D Smith, Mikio Yanagisawa, et al. 2007. *Pre-crash scenario typology for crash avoidance research.* Technical Report. United States. National Highway Traffic Safety Administration.

[51] OpenAI. 2022. *Introducing ChatGPT.* Retrieved November 23, 2023 from https://openai.com/blog/chatgpt

[52] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1, 1 (2016), 33–55.

[53] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. 2019. GeoScenario: An Open DSL for Autonomous Driving Scenario Representation. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. 287–294.

[54] Sebastian Raschka. 2023. *Finetuning LLMs with LoRA and QLoRA: Insights from Hundreds of Experiments.* Retrieved November 23, 2023 from https://lightning.ai/pages/community/lora-insights/

[55] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7881–7892.

[56] 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.

[57] Baichuan Intelligent Technology. 2023. *Baichuan 2*. Retrieved November 26, 2023 from https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat

[58] 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.

[59] Haoxiang Tian, Guoquan Wu, Jiren Yan, Yan Jiang, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. Generating Critical Test Scenarios for Autonomous Driving Systems via Influential Behavior Patterns. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[60] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.

[61] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *Proceedings of the 8th International Conference on Learning Representations*.

[62] Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, and Hermann Felbinger. 2023. Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 49, 3 (2023), 991–1026.

[63] 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.

[64] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2023. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1860–1875.

[65] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. A survey on scenario-based testing for automated driving systems in high-fidelity simulation. *arXiv preprint arXiv:2112.00964* (2021).

Received 16-DEC-2023; accepted 2024-03-02