FIXDRIVE: Automatically Repairing Autonomous Vehicle Driving Behaviour for $0.08 per Violation

Yang Sun[1] Christopher M. Poskitt[1] Kun Wang[2] Jun Sun[1]

[1]*School of Computing and Information Systems*, *Singapore Management University*, Singapore
yangsun.2020@phdcs.smu.edu.sg, cposkitt@smu.edu.sg, junsun@smu.edu.sg

[2]*State Key Laboratory of Industrial Control Technology*, *Zhejiang University*, Hangzhou, China
kunwang_yml@zju.edu.cn

***Abstract*—Autonomous Vehicles (AVs) are advancing rapidly, with Level-4 AVs already operating in real-world conditions. Current AVs, however, still lag behind human drivers in adaptability and performance, often exhibiting overly conservative behaviours and occasionally violating traffic laws. Existing solutions, such as runtime enforcement, mitigate this by automatically repairing the AV's planned trajectory at runtime, but such approaches lack transparency and should be a measure of last resort. It would be preferable for AV repairs to generalise beyond specific incidents and to be interpretable for users. In this work, we propose FIXDRIVE, a framework that analyses driving records from near-misses or law violations to generate AV driving strategy repairs that reduce the chance of such incidents occurring again. These repairs are captured in μDrive, a high-level domain-specific language for specifying driving behaviours in response to event-based triggers. Implemented for the state-of-the-art autonomous driving system Apollo, FIXDRIVE identifies and visualises critical moments from driving records, then uses a Multimodal Large Language Model (MLLM) with zero-shot learning to generate μDrive programs. We tested FIXDRIVE on various benchmark scenarios, and found that the generated repairs improved the AV's performance with respect to following traffic laws, avoiding collisions, and successfully reaching destinations. Furthermore, the direct costs of repairing an AV—15 minutes of offline analysis and $0.08 per violation—are reasonable in practice.**

***Index Terms*—autonomous vehicles, autonomous driving systems, multimodal large language models, driving compliance**

I. INTRODUCTION

Autonomous Vehicles (AVs) are currently undergoing rapid and promising development. Notably, several Level-4 AVs, which do not require driver intervention, have been successfully deployed in real-world traffic scenarios [1]. Prominent examples include Google Waymo [2], Baidu Apollo [3], and TuSimple [4]. These AVs are capable of performing critical tasks such as perception, trajectory planning, and actuation control. However, despite these advancements, AVs are far from perfect and still lag significantly behind human drivers in terms of performance and adaptability. For instance, AVs sometimes exhibit overly conservative driving behaviour, which can lead to situations where they become stuck on the road [5]. Furthermore, Autonomous Driving Systems (ADSs)—the 'brains' of AVs, responsible for perception, decision-making, and control—can also be overly aggressive and cause accidents under specific conditions [6], [7], [8], [9]. Such behaviours are often easily recognisable and avoidable by human drivers, underscoring the need for significant improvements before AVs can match or surpass human driving capabilities.

Existing work offers two categories of solutions to address these problems. The first category uses rule-based runtime enforcement to correct problematic behaviour directly [10], [11], [12], [13], [14], [15], [16], [17]. For example, when an ADS encounters potential violations of specific property specifications (e.g. traffic laws), one proposed solution, outlined in REDriver [10], uses a gradient-based algorithm to modify the AV's planned trajectory. However, these repairs are limited in scope and lack transparency, since they are very low-level and difficult for users to interpret. Furthermore, they are meant to be a measure of last resort rather than a general correction to driving strategies.

The second category involves learning-based methods, which train ADSs to behave like human drivers using real driving data. These approaches focus on exploring and summarising human driver behaviour patterns to guide the driving modes of ADSs [18], [19], such as imitation learning to replicate expert behaviour and train the ADS to drive in a human-like manner [20], [21], [22], [23], [24]. However, these approaches often fall short due to the difficulty of capturing the nuanced decision-making processes of human drivers from limited data, leading to poor generalisation or overfitting to specific tasks. Consequently, there is a need for an AV driving strategy repair approach that generalises beyond specific incidents and is interpretable for users.

Multimodal Large Language Models (MLLMs) appear to be intelligent and ideally suited for improving ADSs due to their advanced text and image understanding and reasoning capabilities [25], [26], [27]. Trained on massive datasets, MLLMs can interpret and replicate human driving behaviour, thereby making ADS decisions more explainable [28]. Existing works (e.g. [29], [30], [31]) explore utilising MLLMs to replace parts of the ADS, such as perception, planning, and control, thereby making the decision-making logic more understandable. For example, GPT-Driver [30] abstracts the perception and prediction results of the ADS into language tokens, then uses OpenAI GPT 3.5 to directly produce the planned trajectory along with explanations. However, the inherent latencies and uncertainty associated with generative models make it impractical to build an ADS based solely on online MLLMs. Additionally, there is a significant gap

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00216

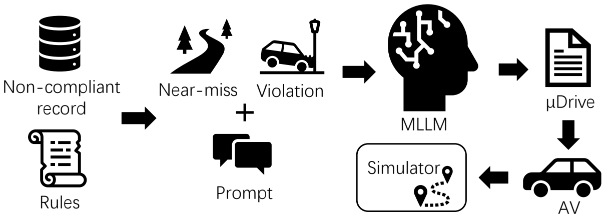


Fig. 1: Overview of FixDrive

between natural language and the actual control commands for autonomous vehicles, making it challenging to directly apply MLLM-generated solutions to real-world driving tasks. Currently, there are no practical approaches that leverage MLLMs in a manner that is both offline and compatible with existing ADS frameworks such as Apollo [3] and Autoware [32].

In this work, we propose FixDrive, a method that analyses records (i.e. comprehensive log files) from bad driving behaviours such as collisions, near misses, or law violations, then generates general AV driving strategy repairs to reduce the chance of such incidents occurring again. Rather than modifying code [33] or applying opaque low-level fixes [10], FixDrive produces repairs in μDrive [34], a high-level Domain-Specific Language (DSL) for specifying the driving behaviours that should occur upon certain triggers (e.g. approaching a traffic light). FixDrive identifies and visualises critical moments from incident records, then utilises an MLLM with zero-shot learning to generate μDrive programs that repair the driving strategy. This translation is executed offline and once per violation, allowing our approach to leverage the reasoning capabilities of MLLMs while mitigating their latency issues. Additionally, by generating repairs in a high-level DSL, they are more interpretable compared to those of low-level gradient-based approaches like REDriver [10]. Note that we categorise methods as either offline or online based on how they interact with runtime driving decisions. For example, an MLLM that generates real-time driving decisions based on the current driving context is classified as an online method. Conversely, FixDrive is an offline framework, enhancing the ADS through repair scripts generated *before* further runs.

An overview of FixDrive is shown in Figure 1. Users need only provide records from executed driving scenarios and the corresponding property specifications (e.g. traffic laws, collision avoidance) that were violated. FixDrive automatically identifies two critical moments from the records (the 'near miss' and the 'violation' moments), visualises them for a multimodal prompt, then utilises a state-of-the-art MLLM (OpenAI GPT 4 [35]) to generate driving strategy repair scripts in the μDrive [34] DSL. Additionally, OpenAI's function calling [36] is used to ensure that the MLLM generates syntactically valid μDrive programs, which are specified via a JSON Schema. The resulting program is then applied to the Apollo ADS [37], dynamically adjusting the parameter settings of the planning module to repair its driving strategy at runtime.

We evaluated FixDrive on a set of benchmark scenarios in which the ADS violated various property specifications, such as different traffic laws, collision avoidance, and successful journey completion. FixDrive provided effective general driving strategy repairs that helped the ADS successfully navigate these problematic scenarios without adversely affecting performance in normal scenarios. Additionally, FixDrive consistently generated effective AV driving strategy repairs with practically reasonable direct costs, i.e. less than 15 minutes (and typically around 10 minutes) of offline analysis and \$0.08 per violation.

$$\phi := \mu \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \,\mathtt{U}_I\, \phi_2$$
$$\mu := f(x_0, x_1, \cdots, x_k) \sim 0 \quad \sim := > \mid \geq \mid < \mid \leq \mid \neq \mid =;$$

Fig. 2: Specification language syntax, where ϕ, ϕ_1 and ϕ_2 are STL formulas, I is an interval, and f is a multivariate linear continuous function over language variables x_i

II. Background and Problem

In this section, we review the architecture of ADSs, DSLs for specifying safety properties and specifying high-level driving behaviours, the current capabilities of MLLMs, and then formally define our problem.

A. Overview of ADSs

State-of-the-art open-source ADSs such as Apollo [37] and Autoware [32] share similar architectures. These systems are typically organised into loosely-coupled modules that communicate via message-passing. Three of these modules are particularly relevant in our context: perception, motion planning, and control.

The perception module receives sensor readings (e.g. from cameras or LiDAR), processes them, and publishes the resulting data to the motion planning module. The motion planning module then classifies the current driving scenario into categories such as *lane follow*, *borrow lane*, and *traffic light handling*. Each scenario has distinct processing logic and key parameters. For example, the *emergency pull-over* scenario involves two key parameters: *Expected_speed* and *Stopping_distance*. During an emergency pull over, the vehicle is expected to rapidly decrease to the *Expected_speed* and then proceed to pull over, with the *Stopping_distance* indicating how far the vehicle should travel before coming to a complete stop. For more detailed information on how these key parameters work, we refer the reader to [37], [32].

For each scenario, the motion planning module generates a corresponding *planned trajectory* based on the map, destination, sensor inputs, and the state of the *ego vehicle* (i.e. the vehicle under ADS control). This planned trajectory outlines the vehicle's future positions at various time points, taking into account the predicted environment, which includes factors such as the anticipated movements of other vehicles (NPCs), pedestrians, and traffic light states. Finally, the control module translates the planned trajectory into control commands (e.g. braking, acceleration, steering, and signaling) so that the ego vehicle follows the planned trajectory, passing through the waypoints with the desired speed, acceleration, steering angle, and gear position.

```
program        ::=  {rule}+
rule           ::=  'rule' string_literal
                    'trigger' event_trigger
                    ['condition' {['!'] condition}+]
                    'then' {action}+
                    ['until' event_trigger]
                    'end'
event_trigger  ::=  event | 'always'
```

Fig. 3: Abstract syntax of μDrive programs

B. *Specifying Safety Properties*

In the context of AVs, safety should not simply mean the absence of collisions, but also adherence to the rules of the road that drivers are supposed to abide by. To that end, we adopt the property specification language used by LawBreaker [8], as well as the project's existing specifications of the traffic laws of China and Singapore. The specification language is based on Signal Temporal Logic (STL), and is evaluated with respect to traces of scenes, providing a way to automatically determine whether a tester-defined property was violated or not in a simulated run of the ADS. We highlight the key features of the specification language below (the full syntax and semantics is given in [8]).

The high-level syntax of the language is shown in Figure 2. A time interval I is of the form $[l, u]$, where l and u are respectively the lower and upper bounds of the interval. Following convention, we write $\Diamond_I\ \phi$ to denote $true\ \mathtt{U}_I\ \phi$; and $\Box_I\ \phi$ to denote $\neg\ \Diamond_I\ \neg\phi$. Intuitively, $\mathtt{U}$, $\Box$, and $\Diamond$ are modal operators that are respectively interpreted as 'until', 'always', and 'eventually'. We omit the time interval when it is $[0, \infty]$.

In general, μ can be regarded as a proposition of the form $f(x_0, x_1, \cdots, x_k) \sim 0$, where f is a multivariate function and x_i for all i in $[0, k]$ is a variable supported in the language.

Example II.1. *Suppose we have a signal variable speed* $= \langle speed(0), speed(1), \ldots, speed(n)\rangle$, *which represents the autonomous vehicle's speed throughout its journey. Then, we can create a simple Boolean Expression* $\mu = speed(t) < 60$ *to test whether the speed of the vehicle is larger than* $60km/h$*. Note that* μ *can be regarded as a proposition of the form* $60 - speed(t) > 0$ *or* $speed(t) - 60 < 0$*. To verify whether* μ *holds true at all time steps, we can use the temporal logic symbol 'always', resulting in the formula* $\varphi = \Box(speed < 60)$.

A specification is evaluated with respect to a trace π of *scenes*, denoted as $\pi = \langle \pi_0, \pi_1, \pi_2 \ldots, \pi_n\rangle$, where each scene π_i is a valuation of the propositions at time step i, and π_0 reflects the state at the start of a simulation. The language follows the standard semantics of STL (see e.g. [38]).

C. *Specifying Driving Behaviours*

The default output of a large language model (LLM) is natural language, which can be vague and challenging to utilise. To obtain specific and actionable behaviours for AVs, we need a robust method to ensure that the output of the LLM is always valid and directly applicable to AVs. To achieve this, we utilise the high-level DSL μDrive [34], which allows driving behaviours to be specified in simple rules that are triggered by contextual events (e.g. approaching a traffic light).

```
rule "Drive slowly through a junction when there is
an obstacle."
trigger
    entering_junction
condition
    obstacle_distance_leq(20)
    is_traffic_light(green)
then
    cruise_speed(30)
until
    exiting_junction
end
```

Fig. 4: μDRIVE driving strategy repair example

The abstract syntax of μDrive in EBNF format is shown in Figure 3. A μDrive program contains one or more rules, each consisting of up to five parts: 1) a *name* or description expressed as a string; 2) a *trigger*, which is an event that causes the rule to be applied; 3) zero or more *conditions*, which constrain the application of the rule; 4) one or more *actions*, which are assignments of driving-related variables that are applied for the duration of the rule; 5) at most one *exit trigger*, which is an event that ends the application of the rule.

Intuitively, events represent states monitored by μDrive as the AV drives through its environment. For example, the events `entering_junction` and `exiting_junction` are set to `True` when the AV is entering or exiting a junction, respectively. Conditions specify what must be true of the current environment to allow the rule to be applied. For example, the conditions `is_traffic_light(green)` and `obstacle_distance_leq(20)` indicate that the rule can take effect only when the traffic light ahead is green and the AV is within 20 metres of an obstacle. Actions are tasks executed throughout the duration of a rule application. For example, the action `cruise_speed(30)` sets the default planning speed of the AV to 30 km/h. An overall example of a μDrive program is shown in Figure 4. This program indicates that within a junction, if an obstacle is detected within 20 metres and the traffic light ahead is green, the default planning speed of the AV should be set to 30 km/h. For a detailed introduction to the grammar of μDrive, we refer readers to [34].

D. *Multimodal Large Language Models (MLLMs)*

MLLMs integrate and process multiple types of data, including text, images, audio, and video. These models utilise the capabilities of large-scale neural networks to comprehend and generate content across different modalities, thereby offering more comprehensive and versatile AI functionalities. State-of-the-art MLLMs, such as OpenAI's GPT-4 [35] and Google's PaLM-E [39], exemplify the advancements in this field. GPT-4, for instance, can process textual inputs while simultaneously understanding and interpreting images, enabling it to describe images, answer related questions, and seamlessly integrate visual information with textual content. Similarly, PaLM-E, a large multimodal embodied language model, integrates textual and visual data to enhance its ability to comprehend and interact with the physical environment.

These models are trained on extensive datasets that encompass diverse forms of media, which allows them to acquire a vast amount of general knowledge. This training enables MLLMs to perform a wide variety of tasks, such as generating detailed image captions, providing contextually aware responses, and enhancing search engines with improved understanding of visual queries. The multimodal approach significantly enhances the model's utility, making it capable of tasks beyond the scope of single-modality models. By integrating multiple types of data, MLLMs are advancing the frontier of AI, creating more intuitive and intelligent systems that better mimic human cognition and understanding.

E. The Problem

Assuming the availability of a powerful MLLM, such as ChatGPT, with the capability to comprehend driving conditions and provide appropriate suggestions, the challenge lies in efficiently leveraging this MLLM to analyse records of AV violations and generate driving strategy repairs in μDrive that would prevent such violations occurring again in the future. We formulate our problem as follows:

Definition 1 (Problem Definition)**.** *Given an MLLM, a record α of the AV in a scenario, and a property specification of ADS behaviour φ, suppose that $\alpha \nvDash \varphi$. The objective is to utilise the MLLM to generate μDrive programs based on the combination of α and φ. Let α' denote the resulting record of the AV by replaying the same scenario with the μDrive programs generated by the MLLM applied. The goal is to increase the likelihood of $\alpha' \vDash \varphi$.*

Intuitively, when a scenario is identified in which the AV violates specified properties, we provide relevant information in the prompt to the MLLM to enable it to understand the current situation. The MLLM then offers suggestions in the form of a μDrive program to help ensure such a violation does not occur again. To maintain minimal and interpretable additional control logic, the μDrive programs should be kept as small as possible. This context highlights two critical requirements for our approach: 1) develop a method to automatically provide accurate and relevant information to the MLLM; 2) ensure that the MLLM's suggestions are translated into valid and effective μDrive programs.

III. Our Approach

FixDrive, our framework for obtaining general driving strategy repairs via MLLMs, comprises three main steps. First, *problem localisation*, which identifies when the violation of the specified properties occurred and any near-miss situations. Second, *prompt generation*, which automatically generates the necessary text prompts and visualisations of specific driving conditions. These specific conditions refer to the time steps when property violations and near-miss situations occurred. Finally, *μDrive script generation*, which formats the MLLM's responses into syntactically valid μDrive scripts, ensuring the driving strategy repairs are executed by the ADS.

A. Problem Localisation

One possible way to allow a language model to comprehend a driving incident is to provide it with a complete record, i.e., a structured log file that captures all necessary data to recreate the driving scenario. This log would include detailed information on the driving environment perception, routing details, predictions of other vehicles, pedestrians, and traffic lights, as well as motion planning and control commands. However, processing such comprehensive records is computationally intensive, costly, and time-consuming. Fortunately, in driving scenarios where certain properties are violated, there are always a few key moments that hold significant importance. Identifying these moments allows for a better understanding of the driving scenario, and can also be automated. For example, ACAV [40] reduces the length of driving records by 62.23% based on a causality analysis, and correctly identifies causal events in 93.64% of a set of generated accident records.

In this work, we develop a lightweight approach—based on a quantitative semantics—to identify two critical moments: the *near-miss* and *violation moments*. The violation moment is the point at which a specific property is violated. The near-miss moment occurs a few time steps before the violation, during which the violation is likely but has not yet happened. For example, if the property that the AV should follow is 'avoid collisions with other objects', the violation moment is when the vehicle collides with another object, while the near-miss moment is when the vehicle fails to maintain a safe distance from other vehicles. The logic is straightforward: the violation moment represents the final outcome, whereas the near-miss moment could be the potential cause of the property violation. We explain in the following how to identify them.

Quantitative Semantics. To locate critical moments, we need a method for quantitatively evaluating whether the current trace satisfies a given property. To achieve this, given a property specification φ and a driving record for the ADS, FixDrive first constructs a trace π from the record by evaluating all variables relevant to φ at each time point. An execution trace π is a sequence of scenes, denoted as $\pi = \langle \theta_0, \theta_1, \ldots, \theta_n \rangle$. A scene θ is a tuple of the form $\theta = (f_0, f_1, \ldots, f_x)$, where each f_i is the valuation of a variable. These variables describe the status of the vehicle, traffic signal states, and traffic conditions. For example, variables such as 'isOverTaking', 'junctionAhead(n)', and 'NPCAhead(n)' indicate whether the vehicle is changing lanes, whether there is a junction ahead within *n* metres, and whether there is a vehicle ahead within *n* metres, respectively. For a detailed introduction to these variables, we refer the readers to [8].

Next, FixDrive computes how 'close' the ego vehicle will come to violating φ. To measure how close a trace π is to violating φ, we adopt a quantitative semantics [38], [41], [42] that produces a numerical *robustness* degree.

Definition 2 (Quantitative Semantics)**.** *Given a trace π and a formula φ, the quantitative semantics is defined as the robustness degree $\rho(\varphi, \pi, t)$, computed as follows. Recall that* *propositions μ are of the form $f(x_0, x_1, \cdots, x_k) \sim 0$.*

$$\rho(\mu, \pi, t) = \begin{cases} -\pi_t(f(x_0, x_1, \cdots, x_k)) & \textit{if} \sim \textit{is} \leq \textit{or} < \\ \pi_t(f(x_0, x_1, \cdots, x_k)) & \textit{if} \sim \textit{is} \geq \textit{or} > \\ \mid \pi_t(f(x_0, x_1, \cdots, x_k)) \mid & \textit{if} \sim \textit{is} \neq \\ - \mid \pi_t(f(x_0, x_1, \cdots, x_k)) \mid & \textit{if} \sim \textit{is} = \end{cases}$$

where t is the time step and $\pi_t(e)$ is the valuation of expression e at time t in π.

$$\begin{aligned} \rho(\neg\varphi, \pi, t) &= -\rho(\varphi, \pi, t) \\ \rho(\varphi_1 \wedge \varphi_2, \pi, t) &= \min\{\rho(\varphi_1, \pi, t), \rho(\varphi_2, \pi, t)\} \\ \rho(\varphi_1 \vee \varphi_2, \pi, t) &= \max\{\rho(\varphi_1, \pi, t), \rho(\varphi_2, \pi, t)\} \\ \rho(\varphi_1 \, \mathtt{U}_\mathtt{I} \, \varphi_2, \pi, t) &= \sup_{t_1 \in t + \mathtt{I}} \min\{\rho(\varphi_2, \pi, t_1), \inf_{t_2 \in [t, t_1]} \rho(\varphi_1, \pi, t_2)\} \\ \rho(\Diamond_\mathtt{I}\varphi, \pi, t) &= \sup_{t' \in t + \mathtt{I}} \rho(\varphi, \pi, t') \\ \rho(\Box_\mathtt{I}\varphi, \pi, t) &= \inf_{t' \in t + \mathtt{I}} \rho(\varphi, \pi, t') \\ \rho(\bigcirc\varphi, \pi, t) &= \rho(\varphi, \pi, t + 1) \end{aligned}$$

where $t + I$ is the interval $[l + t, u + t]$ given $I = [l, u]$. □

Note that the smaller $\rho(\varphi, \pi, t)$ is, the closer π is to violating φ. If $\rho(\varphi, \pi, t) \leq 0$, φ is violated. We write $\rho(\varphi, \pi)$ to denote $\rho(\varphi, \pi, 0)$; $\pi \vDash \varphi$ to denote $\rho(\varphi, \pi, t) > 0$; and $\pi \nvDash \varphi$ to denote $\rho(\varphi, \pi, t) \leq 0$. Note that time is discrete in our setting.

Example III.1. *Let $\varphi = \Box(speed < 60)$, i.e. the speed limit is 60km/h. Suppose π is $\langle(speed \mapsto 0, \dots), (speed \mapsto 0.3, \dots), \cdots (speed \mapsto 50, \dots)\rangle$ where the ego vehicle's max speed is 50km/h at the last time step. We have $\rho(\varphi, \pi) = \rho(\varphi, \pi, 0) = min_{t \in [0, |\pi|]}(60 - \pi_t(speed)) = 10$. This means that trace π satisfies φ, and the robustness value is 10.* □

Violation and Near-Miss Moments. With quantitative semantics, we can now introduce the method to locate the *violation moment* and *near-miss moment*. Given a trace $\pi = \langle\pi_0, \cdots, \pi_n\rangle$, let π^k denote the prefix $\langle\pi_0, \cdots, \pi_k\rangle$, where $k \leq n$. Intuitively, π^k represents the first k time steps of the original trace π. For the *violation moment*, we identify the smallest k that satisfies $\rho(\varphi, \pi^k) \leq 0$. For the *near-miss moment*, we adopt a user-customisable threshold δ. We aim to identify a time step k such that $\rho(\varphi, \pi^k) \leq \delta$ and there does not exist a time step l such that $l < k$ and $\rho(\varphi, \pi^l) < \delta$. Note that δ is determined empirically in our evaluation (see Section IV). Intuitively, k is the earliest time step when the robustness value falls below the threshold δ. We identify the time step k using a sequential search, starting from $k = 0$ and incrementing k until we find a k such that $\rho(\varphi, \pi^k) < \delta$.

Example III.2. *Let $\varphi = \Box(speed < 60)$, i.e. the speed limit is 60km/h. Suppose the threshold $\delta = 5$ Suppose π is $\langle\pi_0 = (speed \mapsto 0, \dots), \pi_1 = (speed \mapsto 1, \dots), \cdots \pi_{90} = (speed \mapsto 90, \dots)\rangle$ where the ego vehicle speed is increasing over time steps and the ego vehicle's max speed is 90km/h at the last time step. We have $\rho(\varphi, \pi) = \rho(\varphi, \pi, 0) = min_{t \in [0, |\pi|]}(60 - \pi_t(speed)) = -30$. Hence, the specification is violated. The following are computed in sequence:*

$$\rho(\varphi, \pi^0) = 60, \rho(\varphi, \pi^1) = 59, \cdots, \rho(\varphi, \pi^{55}) = 5, \cdots,$$
$$\rho(\varphi, \pi^{59}) = 1, \rho(\varphi, \pi^{60}) = 0, \rho(\varphi, \pi^{61}) = -1, \cdots$$

To identify the violation moment, we find the smallest time step k where $\rho(\varphi, \pi^k) \leq 0$. In this case, $k = 60$. Similarly, the smallest time step k where $\rho(\varphi, \pi^k) \leq 5$ is $k = 55$. Therefore, the violation moment occurs at time step 60, and the near-miss moment occurs at time step 55. □

B. Prompt Generation

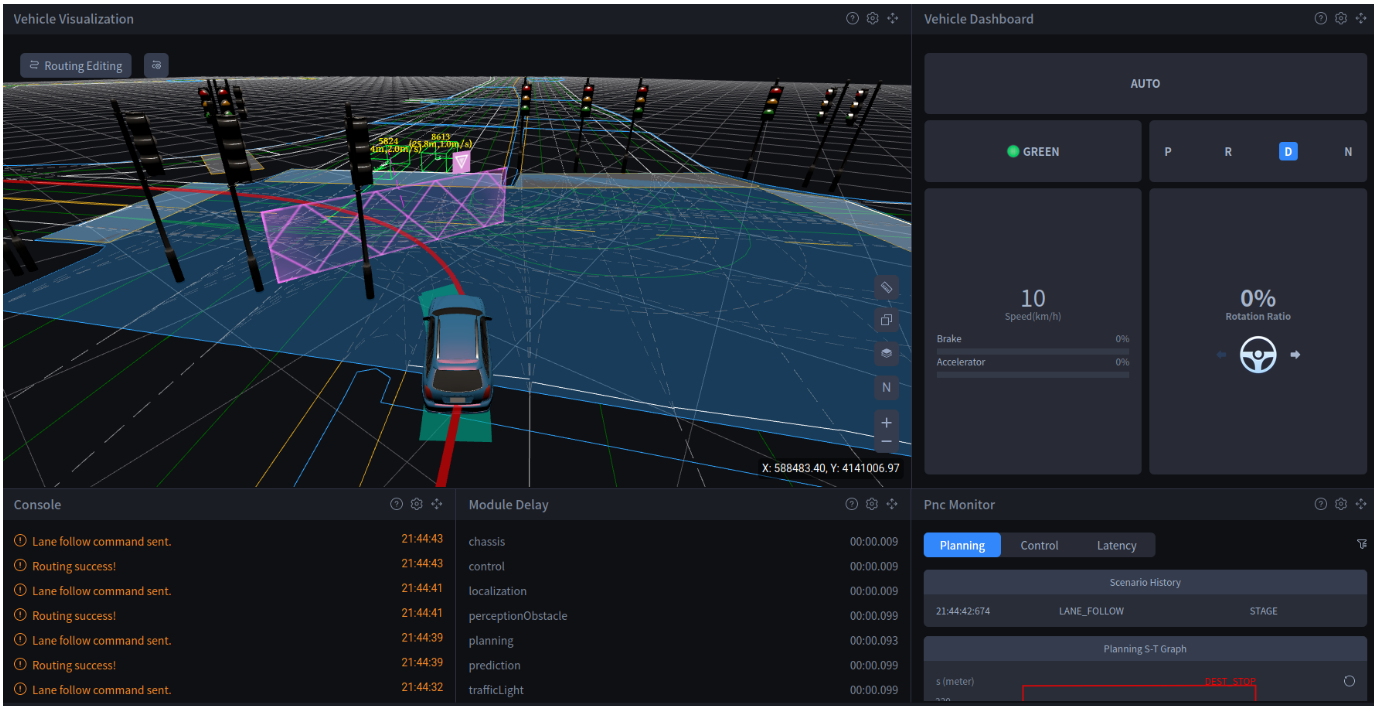
The input for an MLLM can be in various formats, such as images, videos, audio, and text prompts. Given that MLLMs are trained on extensive datasets rich in knowledge, we anticipate they will 'understand' the prompts we provide, much like an intelligent human. In this work, we utilise two types of prompts: visualisations of driving conditions and text descriptions to convey essential information not covered by the visualisations.

Visualisations of Scenarios. In the driving records of ADS trajectories, each time step contains extensive information such as the speed, acceleration, and steering angle of the AV, as well as the positions of other vehicles and pedestrians. While it is possible to describe this information in natural language, it does not provide a direct impression of the driving scenario. For example, given the positions of the AV and another background vehicle, it can be challenging to determine the exact direction of the background vehicle relative to the AV.

Fortunately, visualising the driving scenario can help alleviate this problem, and state-of-the-art ADSs, such as Apollo, offer this capability. Detailed information, including the positions of various objects, can be effectively conveyed through visualisation by displaying a grid map that shows the relative positions of each object.

An example of this visualisation is shown in Figure 5. In this visualisation, the upper-left section, labelled 'Vehicle Visualization', displays the current driving conditions of the AV (marked in blue), other vehicles (marked in green boxes), pedestrians (marked in yellow boxes), cyclists (marked in blue boxes), and unknown objects (marked in purple boxes). Each box includes numerical values indicating the distance to the AV and the current speed of the object. The predicted trajectory of each object is shown as a coloured line. The lower-left section, labelled 'Console', shows logs from the ADS, while the 'Module Delay' section indicates the delay of each module. The right section, labelled 'Vehicle Dashboard', shows the current status of the AV and the detected status of traffic lights ahead. The 'Pnc Monitor' section provides detailed information on the inner decisions of the planning and control modules.

This visualisation compactly encodes rich information in a human-friendly manner. This is important for the transparency of the approach: it allows the MLLM's decisions to be based on the same high-level information that drivers work with, instead of (for example) low-level gradient-based discrepancies.



X: 588483.40, Y: 4141006.97

GREEN

10
Speed(km/h)
Brake 0%
Accelerator 0%

0%
Rotation Ratio

Console

```
Lane follow command sent.    21:44:43
Routing success!             21:44:43
Lane follow command sent.    21:44:41
Routing success!             21:44:41
Lane follow command sent.    21:44:39
Routing success!             21:44:39
Lane follow command sent.    21:44:32
```

Module Delay

```
chassis              00:00.009
control              00:00.009
localization         00:00.009
perceptionObstacle   00:00.099
planning             00:00.093
prediction           00:00.099
trafficLight         00:00.099
```

Scenario History

```
21:44:42:674    LANE_FOLLOW    STAGE
```

Fig. 5: Visualisation of a scenario, which is provided to the MLLM along with an ‘overall prompt’

Overall Prompt. The overall prompt consists of two parts: the first part includes two images illustrating the visualisation of the *violation moment* and the *near-miss moment* as shown in Section III-A, and the second part is a text prompt.

Our text prompts complement the ADS visualisation by providing necessary additional information. These prompts follow a specific workflow to enable automatic generation. First, we specify an identity for the MLLM:

> (identity) Suppose you are a driver.

Next, since the visualisations do not contain weather information, we provide information about the current weather conditions. For example, we state the following if there is no rain, fog, or snow, and the visibility is more than 50 metres:

> (weather) There is nothing noteworthy about the weather.

To aid the MLLM’s understanding of the provided image, we offer background details about the input image. This includes describing different aspects of the visualisation to help the MLLM to understand it:

> (background) In these pictures, the left side shows the visualisation of the driving record. The right side displays the status of the traffic light, vehicle speed, and steering angle. The green boxes indicate detected vehicles, yellow boxes indicate detected pedestrians, blue boxes indicate detected bicycles, and purple boxes indicate unknown objects.

Following this, we describe the property specification that the AV is supposed to satisfy, e.g. the avoidance of collisions or adherence to traffic laws. Note that we use the original traffic laws from [43] as input when the property is an STL-based traffic law specification. For example, if we are testing the property ‘no collisions’, we would state:

> (rule) You are supposed to follow the following rule: Avoid collisions with other objects.

Then, we specify the time gap between the *violation moment* and the *near-miss moment*, and clarify that the former image depicts the violation:

> (sequence) The second picture was taken 4 seconds later than the first picture, capturing the moment when the rule violation occurred.

Finally, we provide some initial settings of the current ADS to assist the MLLM in making decisions:

> (default) In the original ADS, the initial settings are: $max\ planning\ speed = 72km/h$, ...

C. μDrive Script Generation

The driving strategy repairs generated by the MLLM must be in the correct format, i.e. pertaining to the μDrive grammar in Section II-C. This is challenging to achieve with a language model alone, as they have the potential to hallucinate and make mistakes. MLLMs such as ChatGPT-4, however, have added support for function calling [36], which enables users to connect the models with external tools and design integrated workflows. Additionally, OpenAI’s introduction of ‘Structured Outputs’ ensures that the arguments generated by the model adhere precisely to a specified JSON Schema, as defined by the user in the function call.

In FIXDRIVE, we implemented function calling based on a structured JSON Schema that describes the syntax of μDrive programs. This schema guides the MLLM to produce outputs that are always structured as syntactically valid μDrive programs, including all parameters and constructs mandated by the full μDrive grammar. By leveraging this function, we achieve a reliable and structured generation of μDrive scripts that align with expected syntax. For an example of function calling with our JSON Schema, please see our repository [44].

Our function is designed with several key principles in mind: 1) *Structural Integrity:* we ensure that the structure of the output μDrive program always adheres to the syntax of μDrive. Specifically, each program must include one trigger, zero or more conditions, one or more actions, and at most one exit trigger. The sequence is strictly enforced, i.e. trigger, conditions, actions, followed by the exit trigger if it exists. 2) *Comprehensive Descriptions:* to help the MLLM fully understand the meaning and functionality of each element, we add detailed descriptions to all events, conditions, and actions in natural language. These descriptions are sourced from the official documents of ADS and μDrive to ensure accuracy and clarity. 3) *Clear Parameter Definitions:* the unit parameters within events, conditions, and actions are clearly defined to avoid any potential misunderstandings. This precision helps the MLLM to generate accurate and contextually appropriate programs. By adhering to these design principles, integrating μDrive with an MLLM facilitates the creation of actionable and precise programs, that can then be applied in subsequent deployments of the ADS to improve its driving strategy. For more information on our prompts, function calling, and implementation details, please refer to the source code [44].

IV. Implementation and Evaluation

A. Implementation

We have implemented FixDrive for Apollo 9.0 [37] (the latest version at the time of our experiments) and the widely-used MLLM ChatGPT (version ChatGPT 4 Turbo). The simulator utilised in our experiments is the official *Dreamview Plus [45]*, provided by Apollo 9.0.

Our framework, FixDrive, comprises three main components: 1) *Trajectory Record Analysis Tool:* this tool identifies the specific time step at which the quantitative semantics of the trace fall below a specified threshold according to a given specification (such as no collisions and adherence to traffic rules in different countries). It enables us to precisely pinpoint the exact moment a violation occurs and to detect near-miss situations where a violation is likely but has not yet occurred. 2) *Prompt Generator:* this component generates both the text prompt and captures a visualisation of the driving conditions at specific time steps. It organises this information into function calls for ChatGPT. Essentially, the the prompt generator automatically creates the input provided to the MLLM based on specified criteria and recorded data. 3) *Translation and Verification:* the response from the MLLM is translated into a domain-specific language, μDrive, which outlines general driving strategy repairs (e.g. stopping when a pedestrian is ahead). An additional validity check ensures that the syntax is correct. These strategy repairs are then verified using the simulator to ensure they enable the autonomous vehicle to successfully navigate the given scenario.

Our implementation leverages some components from previous work. Specifically, from LawBreaker [8], we utilise its specification language and the corresponding verification algorithm. From μDrive [34], we employ its DSL and backend support for applying new driving strategies in Apollo.

B. Evaluation

Our evaluation considers four Research Questions (RQs):

- **RQ1**: Does FixDrive effectively repair the driving behaviour of an ADS?
- **RQ2**: Are these driving strategy repairs applicable to common driving scenarios?
- **RQ3**: How much effort is required to compute repairs?
- **RQ4**: What is the impact of using images for critical moments instead of text?

RQ1 considers whether FixDrive achieves its primary goal of being able to utilise MLLMs to effectively repair the driving behaviour of an ADS following a violation event. RQ2 investigates the effectiveness of FixDrive's driving strategy repairs, based on a small sample of records, in improving AV performance across different scenarios. RQ3 examines the computational effort needed to utilise FixDrive. RQ4 validates the effects of using images in the prompt instead of ground-truth values in textual prompts. Our experiments utilise both Apollo 9.0 and the Apollo Simulation Platform, referred to as Apollo Studio [46]. To account for simulator randomness (e.g. due to concurrency) each experiment is repeated 20 times, and we present the averages. We utilise a Linux machine (Ubuntu 20.04.5 LTS) with 32GB of RAM, an Intel i7-10700k CPU, and an RTX 2080Ti graphics card.

RQ1: Does* FixDrive *effectively repair the driving behaviour of an ADS? To answer this question, we employed a benchmark of scenarios provided by [34] where Apollo consistently violates specifications. Table I reports the effectiveness of our approach in preventing these violations compared to the original Apollo and the runtime enforcement method REDriver [10]. Note that the threshold for FixDrive is set to 15, determined by an empirical experiment discussed later in this section. The 'Law' column in the table denotes the specific property specification under which the AV is tested. We adopted the formalisation of traffic laws reported in [8] as part of our property specifications and evaluated whether FixDrive can be applied so that the ADS follows them. Specifically, we adopted four rules sourced from the *Regulations for the Implementation of the Road Traffic Safety Law of the People's Republic of China* [43]: *Law38*, *Law44*, *Law46*, and *Law53*. These rules encompass regulations concerning traffic lights (yellow, green, red), speed limits for the fast lane, speed limits under adverse weather conditions (such as fog, rain, and snow), and managing traffic jam, respectively. Additionally, we applied the property specifications 'no collision' and 'finish journey' to evaluate the AV as well. The detailed property specifications of these two rules can be expressed as:

$$no\_collision \equiv \Box(\neg NearestNPC(0.1))$$
$$finish\_journey \equiv \Box(\Diamond_{[0,200]}(speed > 0.5) \vee dest(5))$$

Here, the specification 'no collision' requires that the distance to other objects always be greater than 0.1 metres

TABLE I: Performance comparison of FixDrive, REDriver, and Apollo

Law	Scene	Driver	Fix	Pass	Robustness	Context
no collision	S1	Apollo	-	0%	-0.1	The AV entered the intersection during a green light vehicles, but failed to yield to the straight-moving resulting in an accident.
		REDriver	-	0%	-0.1	
		FixDrive	30%	**100%**	1.37	
	S2	Apollo	-	0%	-0.1	The AV fail to yield to the oncoming straight-through traffic at the stop sign and proceed to make a left turn at the intersection, resulting in an accident.
		REDriver	-	0%	-0.1	
		FixDrive	50%	**100%**	4.41	
Law38	S3	Apollo	-	20%	10.7, 0.0, 0.0	The AV started and entered the intersection when the traffic light was yellow.
		REDriver	-	60%	11.55, 0.5, 0.5	
		FixDrive	45%	**100%**	12.97, 0.5, 0.48	
	S4	Apollo	-	0%	4.7, 0.5, 0.0	The AV entered the intersection on a red light.
		REDriver	-	85%	11.52, 0.5, 0.5	
		FixDrive	100%	**100%**	2.84, 0.5, 0.5	
Law44	S5	Apollo	-	20%	-19.98	The AV is traveling in the fast lane and come to a stop due to an static obstacle (failure to change lanes to an available lane on the right), ultimately failing to reach its destination.
		REDriver	-	**100%**	4.34	
		FixDrive	20%	**100%**	9.58	
Law46	S6	Apollo	-	0%	0.0, -0.2	The AV continues to travel at speeds exceeding 30 kilometres per hour despite fog or rain.
		REDriver	-	**100%**	1.00, 1.00	
		FixDrive	100%	**100%**	1.23, 1.23	
Law53	S7	Apollo	-	0%	0.0	The AV is approaching a junction with traffic jam.
		REDriver	-	0%	0.0	
		FixDrive	50%	**100%**	1.0	
finish journey	S8	Apollo	-	0%	-0.42	The AV failed to overtake a stationary vehicle ahead and became stuck.
		REDriver	-	0%	-0.43	
		FixDrive	15%	**100%**	5.17	

(i.e. $\neg NearestNPC(0.1)$). The specification 'finish journey' requires that the AV must not stop on the road (i.e. $\Diamond_{[0,200]}(speed > 0.5)$), unless it is close to the destination (i.e. $dest(5)$). We refer the readers to our repository in [44] to see all the detailed specifications. The 'Fix' column in Table I shows the proportion of successful driving strategy repairs generated by FixDrive. A driving strategy repair is deemed successful only if it ensures that the AV causes no violations. To evaluate this, we repeat the generation process 20 times for each driving record, generating one unique driving suggestion per run. Each suggestion is then applied to the autonomous vehicle and tested for effectiveness in the same scenario through simulation, allowing us to assess whether the repair successfully resolves the failure. Our empirical analysis indicates that while different suggestions may be generated for the same record, they typically converge into a limited set of outcomes. Consequently, 20 repetitions are empirically sufficient to capture all possible outcomes. The 'Pass' column indicates the proportion of runs that comply with traffic rules. It signifies the success rate of each effective suggestion, which is always 100% for FixDrive. Note that the MLLM generates varied suggestions across runs due to inherent uncertainty. However, empirical analysis shows that most effective suggestions are consistent across trials. Therefore, we select the most frequently successful suggestion to determine the final values in the 'Pass' and 'Robustness' column. The 'Robustness' column in the table illustrates the robustness of the AVs regarding current traffic regulations. The robustness value is calculated as the average performance of these effective suggestions. For example, the three values for the specification *Law38* indicate the robustness values for green light, yellow light, and red light related traffic laws, respectively. Specifically, the robustness value measures how closely the vehicle trajectory adheres to these rules. A higher robustness value indicates a lower likelihood of violating traffic regulations, whereas a lower value indicates a higher likelihood of imminent violation. A value less than or equal to 0 indicates a violation of the corresponding traffic rule. Furthermore, if a regulation comprises multiple sub-rules, the robustness for each sub-rule is sequentially presented.

```
rule "S1 rule1"
trigger
    always
condition
    front_vehicle_closer_than(10)
then
    follow_dist(10)
    yield_dist(15)
    overtake_dist(20)
    obstacle_stop_dist(10)
    obstacle_decrease_ratio(1)
end
```

```
rule "S1 rule2"
trigger
    always
condition
    is_traffic_light(red)
    traffic_light_distance_leq(10)
then
    traffic_light_stop_dist(5)
end
```

Fig. 6: μDrive driving strategy repair scripts for S1

As shown in Table I, Apollo's success rate in these scenarios consistently remains below 50%, often reaching 0%. While REDriver can prevent some of the violations sometimes, there are still instances it cannot address. In contrast, the repairs by FixDrive enable the AV to completely avoid accidents and violations. This is because REDriver focuses on a narrow case-by-case view, making decisions based only on current perception and prediction, and thus heavily depends on the accuracy of the ADS's original predictions, which *may be wrong*. For example, in scenario *S1* where an AV fails to yield to a straight-moving vehicle, REDriver cannot prevent the violation because it cannot reverse the decision 'not to yield', which was based on the prediction that the vehicle would not obstruct its path. In contrast, in the driving strategy repairs generated by FixDrive, as shown in Figure 6, the first program dynamically adjusts parameters such as follow distance, yield distance, overtake distance, stop distance, and obstacle response rate when a vehicle is within 10 metres ahead. Additionally, the second program ensures safety by adjusting the AV's stop distance when approaching a red light and the distance to the stop line is less than 10 metres. This adaptive approach, based on a global perspective, continuously modifies the vehicle's driving style as conditions change, thereby effectively avoiding accidents before they

TABLE II: Effectiveness of FIXDRIVE across varying δ

δ	S1	S2	S3	S4	S5	S6	S7	S8
1	×	×	×	✓	✓	✓	✓	×
5	×	×	✓	✓	✓	✓	✓	×
10	×	✓	✓	✓	✓	✓	✓	✓
15	✓	✓	✓	✓	✓	✓	✓	✓
20	×	✓	✓	✓	✓	✓	✓	×
25	×	✓	✓	✓	✓	✓	✓	×
30	×	✓	✓	✓	✓	✓	✓	×

TABLE III: Performance of FIXDRIVE in official scenarios

Map	Num	Driver	Finish	Accident
Sunnyvale	114	Apollo	108	7
		FIXDRIVE	112	6
SanMateo	103	Apollo	94	1
		FIXDRIVE	95	1
Apollo Virtual	52	Apollo	42	1
		FIXDRIVE	46	1

occur. Due to the inherent uncertainty of the generative model, the proportion of successful repairs sometimes falls below 50% (as shown in 'Fix' column). However, our evaluation of FIXDRIVE's time and cost efficiency in RQ3 demonstrates that generating each driving strategy repair is both time-efficient and cost-effective, mitigating this issue.

To further investigate RQ1, we designed a second experiment to examine how varying FIXDRIVE's thresholds impact its effectiveness. Recall that the threshold defines the fault tolerance of FIXDRIVE, determining what constitutes a near-miss situation, as discussed in Section III-A. The detailed effectiveness evaluation for all thresholds is shown in Table II. In this table, the first column ('δ') denotes the threshold value, ranging from 1 to 30. The subsequent columns represent scenarios S1 to S8 as mentioned above. If FIXDRIVE can provide driving strategy repairs that help the AV resolve the encountered problem, i.e. satisfy the corresponding specification within 10 queries, we mark it with a ✓. Otherwise, we mark it with a ×. As show in Table II, the threshold value significantly impacts FIXDRIVE's effectiveness in certain scenarios. When the threshold δ is very small, such as 1 or 5, the near-miss moments are too close to the violation moment, providing insufficient information about why the violation occurs. Conversely, if δ is too large, such as 30, the near-miss moment may not provide any useful information, as there may be no indication of a potential violation. This can lead to FIXDRIVE's failure to deliver effective programs. For example, in scenario S1, where the AV is making a right turn and fails to yield to vehicles going straight, the timing of the threshold is critical. When the threshold is set to values below 10, the AV is already impeding the straight-moving vehicles at the near-miss moment, making it difficult to implement effective changes. Conversely, when the threshold is set to 20, the AV has not started the right turn yet at the near-miss moment, and the potential problem has not yet arisen. In some scenarios, FIXDRIVE can be effective for all threshold values if the near-miss moment does not provide critical information. For example, in scenario S6, where the vehicle exceeds 30 km/h in snowy conditions, the critical information is that the AV exceeds 30 km/h at the violation moment. Since snow conditions remain consistent, the choice of near-miss moment does not affect FIXDRIVE's effectiveness.

RQ2: Are these driving strategy repairs applicable to common driving scenarios? To answer this question, we applied all the μDrive driving repair scripts generated by FIXDRIVE across the eight scenarios mentioned above. In total, there are 22 different driving strategy repair programs. For detailed information on these driving strategy repair programs, we refer readers to [44]. We applied these strategy repairs to all the official scenarios provided by Apollo across three maps: *Sunnyvale*, *San Mateo*, and *Apollo Virtual*. For the *Sunnyvale* map, there are 114 different scenarios. The *San Mateo* map contains 103 different scenarios, while the *Apollo Virtual* map includes 52 different scenarios. These scenarios cover most situations encountered during daily city driving, such as passing traffic lights, yielding to pedestrians and priority vehicles, cutting in, changing lanes, overtaking, and making U-turns. For detailed descriptions of these official scenarios provided by Apollo, refer to [46].

First, we compared Apollo with FIXDRIVE (i.e. Apollo with the driving repairs applied) regarding *Completion Rate* and *Accidents*, as shown in Table III. Regarding *Completion Rate*, we evaluated whether the AV successfully reached the destination and completed the journey, as indicated in the 'Finish' column. All scenarios completed by Apollo were also completed by FIXDRIVE. Additionally, FIXDRIVE could finish extra scenarios where Apollo got stuck. For example, in some scenarios, Apollo failed to overtake a stationary vehicle ahead because it followed too closely, while FIXDRIVE completed these scenarios by maintaining a larger following distance. Regarding *Accidents*, we examined the number of accidents caused by Apollo and FIXDRIVE, as shown in the 'Accident' column. FIXDRIVE not only avoided causing new accidents but also prevented an accident in one scenario. It is important to note that all remaining accidents were not caused by the AV. They were caused by 'irrational' vehicles or pedestrians colliding with the driver from behind. Typically, this occurred when the AV had reached its destination and stopped, and another vehicle hit it from the back.

To further investigate RQ2, we conducted an in-depth analysis. Specifically, for scenarios that Apollo and FIXDRIVE successfully completed, we analysed various aspects of the trajectories, including the speed and acceleration at each point, the distance to the nearest obstacle, the total duration of vehicle stops, and the energy consumption, as shown in Table IV. Regarding *Speed and Acceleration*, we compared the speed and acceleration between Apollo and FIXDRIVE, as shown in the 'Speed(m/s)' and 'Acceleration(m/s^2)' columns. Both the average and maximum speeds were examined, with the values in the table representing the averages across all scenarios. Overall, the AV operated at slower speeds under Apollo compared to FIXDRIVE. However, this does not imply that FIXDRIVE drives more aggressively than Apollo. In fact, FIXDRIVE only increased its speed when there were no other vehicles or pedestrians nearby, ensuring safe and considerate driving behaviour. Regarding *Obstacle Distance*, we examined

TABLE IV: Performance comparison of FIXDRIVE and Apollo in official scenarios

Map	Num	Driver	Speed(m/s)		Accelerate(m/s^2)		Obstacle Distance(m)		Stop Time(s)	Energy(J)
			Ave	Max	Ave	Max	Ave	Min		
Sunnyvale	108	Apollo	3.31	7.86	0.38	1.91	51.43	6.74	15.56	132840.52
		FIXDRIVE	3.76	8.29	0.46	2.19	51.71	6.90	12.63	147434.25
SanMateo	94	Apollo	2.58	6.28	0.41	1.47	31.27	6.85	6.95	70570.81
		FIXDRIVE	2.84	6.20	0.44	1.80	30.97	7.45	4.48	69660.75
Apollo Virtual	42	Apollo	4.10	8.43	0.43	1.75	78.48	4.06	13.95	157633.50
		FIXDRIVE	4.59	9.18	0.50	1.95	78.22	4.39	6.00	210024.93

the average and maximum distances to other objects, as shown in the 'Obstacle Distance(m)' column. The results indicated that FIXDRIVE maintained a greater distance from other vehicles despite its higher speed, demonstrating both efficiency and safety. Regarding *Stop Time*, we examined the time that the AV stopped on the road, as shown in the 'Stop Time(s)' column. The results indicated that FIXDRIVE had less stop time than Apollo, suggesting a smoother driving experience. Regarding *Energy*, we provided a rough estimate of the average energy consumption for Apollo and FIXDRIVE. The energy was calculated using the formula: $\sum_{t=1}^{n-1} \frac{1}{2} m(v_{t+1}^2 - v_t^2)$, where m is the vehicle's mass (1500 kg), n is the length of the trace, and v_t is the speed of the AV at time step t. This formula measures the energy consumption based on changes in speed. FIXDRIVE consumed more energy than Apollo because it generally travelled at higher speeds, which involved more frequent acceleration and deceleration processes.

Based on these detailed checks, we conclude that the driving strategy repairs provided by FIXDRIVE not only promote smoother driving but also contribute to fewer accidents, underscoring its suitability for various common driving scenarios.

TABLE V: Computational effort required by FIXDRIVE

step		S1	S2	S3	S4	S5	S6	S7	S8
trace		329s	351s	281s	127s	61s	63s	395s	529s
localisation		187s	200s	167s	156s	134s	155s	205	168
prompt		0.22s	0.24s	0.21s	0.23s	0.16s	0.17s	0.16s	0.22
query	time	10.6s	11.4s	14.1s	8.9s	14.9s	8.6s	23.3s	10.8s
	input	7352	7352	7436	7435	7508	7498	7504	7350
	output	179	163	121	185	97	81	123	82
overall time		527s	563s	462s	292s	210s	227s	624s	708s
cost($)		0.079	0.078	0.078	0.080	0.078	0.077	0.079	0.076

RQ3: How much effort is required to compute repairs? To answer this question, we present a detailed breakdown of time and token consumption (using model ChatGPT 4 turbo) for FIXDRIVE, as shown in Table V. The 'step' column lists all necessary steps for FIXDRIVE to generate driving strategy repairs. The 'trace' step involves converting a given record into a trace. The 'localisation' refers to identifying *near-miss* and *violation* moments. The 'prompt' involves automatically generating prompts for the LLM input, while 'query' denotes querying the LLM for a response.

We detail the time consumption for each step, all measured in seconds. For the whole process, the most time-consuming steps involve two parts: trace generation and moment localisation. Trace generation takes a few minutes due to the thousands of time steps within a trace, typically about one hundred time steps per second. Moment localisation involves calculating the robustness value multiple times, resulting in relatively high time consumption. However, since our method is offline, trace generation and moment localisation need to be performed only once per test case, making the process efficient for practical use. For each test case, the whole process typically takes around 10 minutes to perform, always below 15 minutes, on a desktop with 32GB of RAM, an Intel i7-10700k CPU, and an RTX 2080Ti graphics card, which is a manageable effort.

Additionally, we measure the number of tokens required for querying the LLM. The 'input' and 'output' rows in the Table V indicate the average number of input and output response tokens for ChatGPT 4 turbo. The number of tokens, including those for images, is calculated using ChatGPT's official tool [47]. At the time of experimentation, the direct cost for 1 million input prompt tokens was $10, while 1 million output response tokens cost $30. This indicates that each driving suggestion costs less than $0.08, making it affordable, as shown in the last row of the table.

TABLE VI: Comparison of FIXDRIVE and a text-only method

Scene	performance		input token		output token		cost($)	
	ours	text	ours	text	ours	text	ours	text
S1	30%	5%	7352	8370	179	308	0.079	0.092
S2	50%	0%	7352	7294	163	286	0.078	0.082
S3	45%	5%	7436	8451	121	245	0.078	0.092
S4	100%	15%	7435	7741	185	247	0.080	0.085
S5	20%	0%	7508	8442	97	294	0.078	0.093
S6	100%	100%	7498	7433	81	189	0.077	0.080
S7	50%	20%	7504	10978	123	351	0.079	0.120
S8	15%	0%	7350	7240	82	241	0.076	0.080

RQ4: What is the impact of using images for critical moments instead of text? FIXDRIVE utilises visualisations of violation and near-miss moments as part of its prompt for the MLLM. But what would happen if we described these scenarios using only textual prompts instead? To explore this question, we establish a text-only prompt-based method as our baseline. To ensure a fair comparison, we keep all other design elements consistent with FIXDRIVE, except that descriptions of the violation and near-miss moments are provided solely in text. We extract key information from records following the LawBreaker methodology [8], crafting detailed descriptions for each variable to ensure clarity. These descriptions are formatted and refined using ChatGPT-4 Turbo, optimising them for MLLM interpretation. An example prompt is available in our repository for reference [44].

Table VI compares FIXDRIVE and the text-only method in terms of performance, input/output token usage, and cost. The performance threshold, set at 15 based on empirical experimentation (discussed in RQ1), includes 20 repetitions per scenario. The 'Performance' column shows the proportion of successful driving strategy repairs generated by FIXDRIVE (referred to as 'ours') and the text-only method (referred to as 'text'). Here, success indicates that the AV correctly follows the traffic rule after the repair. The 'Input/Output Tokens' columns display the average input and output token counts per query. As shown, using images significantly enhances performance while reducing costs per query. Images effectively convey spatial details that are challenging to capture in text yet are easily processed by the vision modality. Interestingly, image prompts consume fewer input tokens than detailed text descriptions. Moreover, text-heavy prompts often result in more output tokens, suggesting that the MLLM is more prone to generating extraneous driving strategy repairs when overloaded with extensive text inputs.

Threats to Validity. The inherent randomness of generative models and the limitations of the original ADS introduce threats to validity. First, FIXDRIVE cannot guarantee the effectiveness of every generated suggestion. To mitigate this, we generate 20 driving strategy repairs per case and evaluate them in an AV simulator, leveraging fast and cost-effective querying for robustness.

Some scenarios remain beyond FIXDRIVE's full control, such as rear-end collisions, where following distance depends on the trailing vehicle. While risk-reduction measures exist, complete prevention is challenging. Moreover, FIXDRIVE may propose valid repairs that prove ineffective due to ADS design constraints. For instance, Apollo's overly cautious behaviour might prevent overtaking, even when FIXDRIVE suggests it. Refining text prompts can help address such issues.

Integrating μDrive into an ADS poses challenges, but once incorporated, it streamlines further modifications, enabling efficient system refinement through various μDrive scripts.

V. RELATED WORK

AVs have been the subject of extensive research in recent years, leading to significant advancements in their capabilities. Early efforts in AV development focused on improving core functionalities such as perception, planning, and control [48], [49]. These areas are critical for enabling AVs to navigate complex environments safely. However, the limitations of AVs compared to human drivers, particularly regarding adaptability and decision-making in unpredictable scenarios, have prompted further research into more intelligent systems.

Several approaches have been proposed to address challenges encountered by AVs at runtime. Rule-based systems have been used to ensure adherence to safety and traffic regulations. For example, runtime enforcement mechanisms monitor the vehicle's actions [15], [16], [17] to prevent collisions and other unsafe behaviours [11], [12], [13], [14]. Similarly, gradient-based algorithms, such as those in REDriver [10], offer real-time solutions for handling property specification violations. While these methods provide valuable safety nets, their utility is limited by their narrower focus on specific tasks.

Recognising the expertise of human drivers, researchers have explored various ways to model and replicate human driving behaviour in AVs. Imitation learning has emerged as a prominent technique for training AVs to mimic expert human drivers [20], [21], [23]. These methods aim to capture the nuanced decision-making processes of human drivers to improve AV performance. However, challenges such as limited training data and the complexity of human driving behaviour have hindered the generalisation of these approaches [24]. As a result, there is a growing interest in developing more advanced systems that can better bridge the gap between human intelligence and AV technology.

The advent of MLLMs has opened new avenues for enhancing AV intelligence. MLLMs, with their advanced text and image understanding capabilities, offer promising solutions for interpreting and replicating human driving behaviour. They can provide natural language explanations for their decisions, thereby enhancing transparency and trust [28]. Existing research has explored the use of MLLMs in various components of AVs, including perception, planning, and control [29], [30], [31]. For instance, LLM-Driver [29] abstracts driving scenarios into 2D object-level vectors and directly applies the LLM output as control commands for the AV system. GPT-Driver [30] translates motion planner inputs and outputs into language tokens, utilising LLMs as motion planners for AVs. Wen et al. [31] evaluated the potential of ChatGPT-4 as an autonomous driving agent, demonstrating its advanced scene understanding and causal reasoning capabilities. These works primarily employ LLMs for object perception, motion planning, and actuation control within AV systems. Despite their potential, the inherent delays and uncertainties associated with generative models pose challenges for real-time AV operations. Additionally, the gap between natural language and control commands remains a significant hurdle.

FIXDRIVE, in contrast, is a framework that generates driving strategy repairs for AVs. By providing a general offline solution, it ensures that MLLMs can generate general driving suggestions that are directly applicable to AVs. Through comprehensive testing in various scenarios, FIXDRIVE has demonstrated its effectiveness in improving AV decision-making and adherence to property specifications, offering an advancement to the field of autonomous driving.

VI. CONCLUSION

We have proposed FIXDRIVE, a framework that uses MLLMs to enhance ADSs by generating intelligent driving strategy repairs. FIXDRIVE identifies critical moments in driving scenarios and generates prompts to ensure MLLMs produce valid suggestions in a DSL, μDrive, for direct application in an ADS. Experimental results show that FIXDRIVE improves ADS performance in various challenging scenarios, providing efficient and cost-effective driving suggestions. This framework represents a step towards bridging the gap between human expertise and automated driving technology, enhancing the adaptability and reliability of AVs.

ACKNOWLEDGMENT

This research is supported by the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (Award ID: MOET32020-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

References

[1] SAE On-Road Automated Vehicle Standards Committee, “Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles,” *SAE International: Warrendale, PA, USA*, 2021.

[2] Waymo. (2025) Waymo Driver. https://waymo.com/waymo-driver/. Online; accessed Feb 2025.

[3] Baidu. (2025) Apollo. https://www.apollo.auto/. Online; accessed Feb 2025.

[4] TuSimple. (2024) Autonomous driving technology designed for trucks. https://www.tusimple.com/technology/. Online; accessed Nov 2024.

[5] Z. Wan, J. Shen, J. Chuang, X. Xia, J. Garcia, J. Ma, and Q. A. Chen, “Too afraid to drive: Systematic discovery of semantic DoS vulnerability in autonomous driving planning under physical-world attacks,” in *NDSS*. The Internet Society, 2022.

[6] S. K. Bashetty, H. B. Amor, and G. Fainekos, “DeepCrashTest: Turning dashcam videos into virtual crash tests for automated driving systems,” in *ICRA*. IEEE, 2020, pp. 11 353–11 360.

[7] G. Li, Y. Li, S. Jha, T. Tsai, M. B. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. K. Iyer, “AV-FUZZER: Finding safety violations in autonomous driving systems,” in *ISSRE*. IEEE, 2020, pp. 25–36.

[8] Y. Sun, C. M. Poskitt, J. Sun, Y. Chen, and Z. Yang, “LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles,” in *ASE*. ACM, 2022, pp. 62:1–62:12.

[9] Y. Zhou, Y. Sun, Y. Tang, Y. Chen, J. Sun, C. M. Poskitt, Y. Liu, and Z. Yang, “Specification-based autonomous driving system testing,” *IEEE Trans. Software Eng.*, vol. 49, no. 6, pp. 3391–3410, 2023.

[10] Y. Sun, C. M. Poskitt, X. Zhang, and J. Sun, “REDriver: Runtime enforcement for autonomous vehicles,” in *ICSE*. ACM, 2024, pp. 176:1–176:12.

[11] J. Grieser, M. Zhang, T. Warnecke, and A. Rausch, “Assuring the safety of end-to-end learning-based autonomous driving through runtime monitoring,” in *DSD*. IEEE, 2020, pp. 476–483.

[12] D. K. Hong, J. Kloosterman, Y. Jin, Y. Cao, Q. A. Chen, S. A. Mahlke, and Z. M. Mao, “AVGuardian: Detecting and mitigating publish-subscribe overprivilege for autonomous vehicle systems,” in *EuroS&P*. IEEE, 2020, pp. 445–459.

[13] K. Cheng, Y. Zhou, B. Chen, R. Wang, Y. Bai, and Y. Liu, “Guardauto: A decentralized runtime protection system for autonomous driving,” *IEEE Trans. Computers*, vol. 70, no. 10, pp. 1569–1581, 2021.

[14] S. Shankar, U. V. R, S. Pinisetty, and P. S. Roop, “Formal runtime monitoring approaches for autonomous vehicles,” in *OVERLAY’20*, ser. CEUR Workshop Proceedings, vol. 2785. CEUR-WS.org, 2020, pp. 89–94.

[15] M. Mauritz, F. Howar, and A. Rausch, “Assuring the safety of advanced driver assistance systems through a combination of simulation and runtime monitoring,” in *ISoLA (2)*, ser. LNCS, vol. 9953, 2016, pp. 672–687.

[16] B. D’Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, “LOLA: Runtime monitoring of synchronous systems,” in *TIME*. IEEE Computer Society, 2005, pp. 166–174.

[17] K. Watanabe, E. Kang, C. Lin, and S. Shiraishi, “Runtime monitoring for safety of intelligent vehicles,” in *DAC*. ACM, 2018, pp. 31:1–31:6.

[18] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, “End-to-end autonomous driving: Challenges and frontiers,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 12, pp. 10 164–10 183, 2024.

[19] A. Prakash, K. Chitta, and A. Geiger, “Multi-modal fusion transformer for end-to-end autonomous driving,” in *CVPR*. Computer Vision Foundation / IEEE, 2021, pp. 7077–7087.

[20] K. Sama, Y. Morales, H. Liu, N. Akai, A. Carballo, E. Takeuchi, and K. Takeda, “Extracting human-like driving behaviors from expert driver data using deep learning,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 9315–9329, 2020.

[21] J. Wei, J. M. Dolan, and B. Litkouhi, “A learning-based autonomous driver: Emulate human driver’s intelligence in low-speed car following,” in *Unattended Ground, Sea, and Air Sensor Technologies and Applications XII*, vol. 7693. SPIE, 2010, pp. 93–104.

[22] L. Xu, J. Hu, H. Jiang, and W. Meng, “Establishing style-oriented driver models by imitating human driving behaviors,” *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2522–2530, 2015.

[23] D. Xu, Z. Ding, X. He, H. Zhao, M. Moze, F. Aioun, and F. Guillemard, “Learning from naturalistic driving data for human-like autonomous highway driving,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 12, pp. 7341–7354, 2021.

[24] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, “A survey on imitation learning techniques for end-to-end autonomous vehicles,” *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 14 128–14 147, 2022.

[25] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *NeurIPS*, 2020.

[26] OpenAI, “GPT-4 technical report,” *CoRR*, vol. abs/2303.08774, 2023.

[27] S. Yin, C. Fu, S. Zhao, K. Li, X. Sun, T. Xu, and E. Chen, “A survey on multimodal large language models,” *CoRR*, vol. abs/2306.13549, 2023.

[28] C. Cui, Y. Ma, X. Cao, W. Ye, Y. Zhou, K. Liang, J. Chen, J. Lu, Z. Yang, K. Liao, T. Gao, E. Li, K. Tang, Z. Cao, T. Zhou, A. Liu, X. Yan, S. Mei, J. Cao, Z. Wang, and C. Zheng, “A survey on multimodal large language models for autonomous driving,” in *WACV (Workshops)*. IEEE, 2024, pp. 958–979.

[29] L. Chen, O. Sinavski, J. Hünermann, A. Karnsund, A. J. Willmott, D. Birch, D. Maund, and J. Shotton, “Driving with LLMs: Fusing object-level vector modality for explainable autonomous driving,” in *ICRA*. IEEE, 2024, pp. 14 093–14 100.

[30] J. Mao, Y. Qian, H. Zhao, and Y. Wang, “GPT-Driver: Learning to drive with GPT,” *CoRR*, vol. abs/2310.01415, 2023.

[31] L. Wen, X. Yang, D. Fu, X. Wang, P. Cai, X. Li, T. Ma, Y. Li, L. Xu, D. Shang, Z. Zhu, S. Sun, Y. Bai, X. Cai, M. Dou, S. Hu, B. Shi, and Y. Qiao, “On the road with GPT-4V(ision): Early explorations of visual-language model on autonomous driving,” *CoRR*, vol. abs/2311.05332, 2023.

[32] Autoware.AI, “Autoware.AI,” www.autoware.ai/, 2025, online; accessed Feb 2025.

[33] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, “A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each,” in *ICSE*. IEEE Computer Society, 2012, pp. 3–13.

[34] K. Wang, C. M. Poskitt, Y. Sun, J. Sun, J. Wang, P. Cheng, and J. Chen, “μDrive: User-controlled autonomous driving,” *CoRR*, vol. abs/2407.13201, 2024.

[35] OpenAI. (2025) ChatGPT. https://openai.com. Online; accessed Feb 2025.

[36] OpenAI, “Function calling,” https://platform.openai.com/docs/guides/function-calling, 2025, online; accessed Feb 2025.

[37] Baidu, “Apollo 9.0,” https://github.com/ApolloAuto/apollo/releases/tag/v9.0.0, 2023, online; accessed Feb 2025.

[38] O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *FORMATS/FTRTFT*, ser. LNCS, vol. 3253. Springer, 2004, pp. 152–166.

[39] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, “PaLM-E: An embodied multimodal language model,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 8469–8488.

[40] H. Sun, C. M. Poskitt, Y. Sun, J. Sun, and Y. Chen, “ACAV: A framework for automatic causality analysis in autonomous vehicle accident recordings,” in *ICSE*. ACM, 2024, pp. 102:1–102:13.

[41] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, “Robust online monitoring of signal temporal logic,” *Formal Methods Syst. Des.*, vol. 51, no. 1, pp. 5–30, 2017.

[42] D. Nickovic and T. Yamaguchi, “RTAMT: Online robustness monitors from STL,” in *ATVA*, ser. LNCS, vol. 12302. Springer, 2020, pp. 564–571.

[43] Chinese Government, “Regulations for the implementation of the road traffic safety law of the People’s Republic of China,” http://www.gov.cn/gongbao/content/2004/content_62772.htm, 2021, online; accessed Feb 2025.

[44] “FixDrive source code & supplementary materials,” 2025. [Online]. Available: https://github.com/lawbreaker2022/FixDrive2025

[45] Baidu, “Dreamview Plus,” https://github.com/ApolloAuto/apollo/tree/master/modules/dreamview_plus, 2024, online; accessed Feb 2025.

[46] Baidu, “Apollo Studio,” https://apollo.baidu.com/workspace, 2025, online; accessed Feb 2025.

[47] OpenAI, “Tokenizer,” https://platform.openai.com/tokenizer, 2025, online; accessed Feb 2025.

[48] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. R. Pratt, M. Sokolsky, G. Stanek, D. M. Stavens, A. Teichman, M. Werling, and S. Thrun, “Towards fully autonomous driving: Systems and algorithms,” in *Intelligent Vehicles Symposium*. IEEE, 2011, pp. 163–168.

[49] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A survey of autonomous driving: Common practices and emerging technologies,” *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.