Autoware.Flex: Human-Instructed Dynamically Reconfigurable Autonomous Driving Systems

Ziwei Song[1], Mingsong Lv[2], Tianchi Ren[1], Chun Jason Xue[3], Jen-Ming Wu[4], Nan Guan[1,*]
[1]City University of Hong Kong, Hong Kong SAR
[2]Hong Kong Polytechnic University, Hong Kong SAR
[3]Mohamed bin Zayed University of Artificial Intelligence, UAE
[4]Hon Hai Research Institute, Taiwan

***Abstract*—Existing Autonomous Driving Systems (ADS) independently make driving decisions, but they face two significant limitations. First, in complex scenarios, ADS may misinterpret the environment and make inappropriate driving decisions. Second, these systems are unable to incorporate human driving preferences in their decision-making processes. This paper proposes Autoware.Flex, a novel ADS system that incorporates human input into the driving process, allowing users to guide the ADS in making more appropriate decisions and ensuring their preferences are satisfied. Achieving this needs to address two key challenges: (1) translating human instructions, expressed in natural language, into a format the ADS can understand, and (2) ensuring these instructions are executed safely and consistently within the ADS' decision-making framework. For the first challenge, we employ a Large Language Model (LLM) assisted by an ADS-specialized knowledge base to enhance domain-specific translation. For the second challenge, we design a validation mechanism to ensure that human instructions result in safe and consistent driving behavior. Experiments conducted on both simulators and a real-world autonomous vehicle demonstrate that Autoware.Flex effectively interprets human instructions and executes them safely.**

I. INTRODUCTION

Existing Autonomous Driving Systems (ADS) independently make driving decisions based on their perception of the environment [15] [3]. While effective in many scenarios, they still face significant limitations.

First, ADS may misinterpret the environment, leading to inappropriate driving decisions in complex scenarios [25] [19] [2]. For example, consider a road intersection where the traffic lights malfunction and remain stuck on a red signal, as shown in Fig. 1. To manage traffic, a police officer temporarily directs vehicles at the intersection. While an ADS might be trained to recognize traffic lights and human figures, it could fail to interpret this special situation. Consequently, the ADS might stop the vehicle and wait for the traffic light to turn green. In contrast, a human driver can easily understand the context and follow the instructions of the traffic officer to cross the intersection.

Second, existing ADS do not consider accommodating user-specific driving preferences [8] [14]. For example, an ADS typically changes lanes and adjusts the vehicle's speed to optimize traffic flow and avoid blockages. However, a user in an autonomous vehicle might prefer to cruise in the outermost lane at a very low speed while searching for the destination on the roadside. In such cases, the ADS, unaware of the user's specific requirements, may drive the vehicle in a manner that is safe but inconsistent with the user's preferences, which can significantly diminish user experience.

Fig. 1. A complex scenario: traffic lights malfunction, and a traffic officer directs vehicles at the intersection.

To address these limitations, we propose *a novel approach that incorporates human input into ADS's decision-making process*. This approach allows users to guide the ADS through complex scenarios, ensuring more appropriate decisions while also satisfying their personal driving preferences.

Achieving this goal presents two key challenges. The first challenge is *translating human instructions, typically expressed in natural language, into a format that ADS can understand*. While natural language is intuitive for users, ADS systems rely on predefined, structured formats to express information specific to autonomous driving tasks [26]. A Large Language Model (LLM) could be used for translation [33] [30]; however, LLMs often lack the domain-specific knowledge required for ADS. To address this, we develop an ADS-specialized knowledge base to provide the LLM with necessary domain-specific information, enabling effective translation.

The second challenge is *ensuring that user instructions are executed safely and consistently within the ADS's original decision-making framework*. User instructions cannot always be assumed to be safe. For example, a user might inadvertently issue a command that leads to unsafe driving behavior, such as

* Corresponding author: Nan Guan, nanguan@cityu.edu.hk.

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00011

requesting a lane change while the vehicle is cruising at a high speed. To address this, we develop a mechanism to validate and safeguard user instructions, ensuring they are only executed when safe driving can be guaranteed. This mechanism resolves potential conflicts between user instructions and the ADS's decisions.

To implement the proposed approach, we develop Autoware.Flex, a new ADS system built on Autoware.Universe [16], the world's leading open-source software for autonomous driving. Experiments are conducted on both a simulation platform (AWSIM) [17] and a real-world autonomous vehicle prototyped by our team. The results demonstrate that Autoware.Flex effectively interprets and safely executes user instructions, significantly enhancing the capabilities of existing ADS.

Additionally, we develop an ADS knowledge base to assist domain-specific language translation by LLMs via the Retrieval-Augmented Generation (RAG) architecture [20]. Our knowledge base extracts the key information relevant to autonomous driving decision-making. Experimental results show that our knowledge base achieves higher accuracy in assisting LLMs than standard domain-specific resources, such as the Autoware manual. We also create a dataset of ground truths, mapping natural language user instructions to corresponding ADS representations. These resources are valuable for advancing research in this area.

II. Autoware.Flex Overview

Autoware.Flex introduces a novel approach to incorporate human input into the decision-making process of an ADS. This allows users to guide the ADS through complex scenarios, enabling more appropriate decisions while also accommodating their personal driving preferences. The architecture of Autoware.Flex is shown in Fig. 2. Autoware.Flex comprises two primary components: Instruction Translation and Instruction Execution, each addressing a key challenge outlined in the introduction.

A. Instruction Translation

The Instruction Translation component processes user instructions provided in natural language and leverages a Large Language Model (LLM) to generate an AutoIR program — a representation that the ADS can understand. An AutoIR program specifies where in the driving loop the user instruction is injected, as well as key parameters.

While LLMs are adept at understanding human language [18] [30] [23], they typically lack domain-specific knowledge of ADS that is essential for generating accurate AutoIR programs. To address this limitation, we construct an ADS-specific knowledge base that assists the LLM via the Retrieval-Augmented Generation (RAG) architecture. This knowledge base provides the domain knowledge required for the LLM to effectively translate natural language instructions into AutoIR representations. The technical details of this component will be provided in Sec. III.

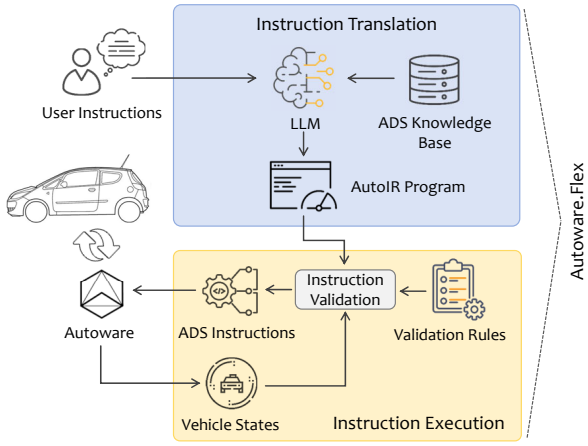


Fig. 2. An overview of Autoware.Flex

B. Instruction Execution

The Instruction Execution component takes AutoIR programs produced by the Instruction Translation module as input and converts them into actionable ADS instructions. These instructions are then injected into the ADS (in this paper, Autoware.Universe) to influence the vehicle's behavior.

A critical function of this component is ensuring that user instructions are executed without compromising safety. This is achieved through a rule-based validation process that evaluates the current vehicle status and the environment. The validation checks whether predefined safety rules — derived from human expertise on how ADS parameters affect driving behavior — are satisfied. If the validation succeeds, the AutoIR program is translated into ADS instructions. This translation step is straightforward and ensures seamless integration with the ADS. The validated instructions are then sent to the ADS for execution, enabling safe and user-guided driving behaviors. The technical details of this component will be provided in Sec. IV.

III. Translating User Instructions

Translating user instructions into AutoIR programs involves two main steps. First, we determine whether a user instruction is relevant to autonomous driving. Only instructions related to driving are processed further. Second, if the user instruction is relevant, it serves as the input to generate the corresponding AutoIR program. The detailed workflow is shown in Fig. 3.

A. Relevance Analysis

When a user is in an autonomous vehicle, their conversations may cover a wide range of topics, many of which might not be related to autonomous driving. To ensure the system processes only relevant input, we need to filter out unrelated dialogue, focusing solely on instructions pertaining to the vehicle's operation.

To achieve this, we leverage the capabilities of LLMs to perform this classification task. Instead of re-training the LLM, we adopt in-context learning, specifically Chain of Thought

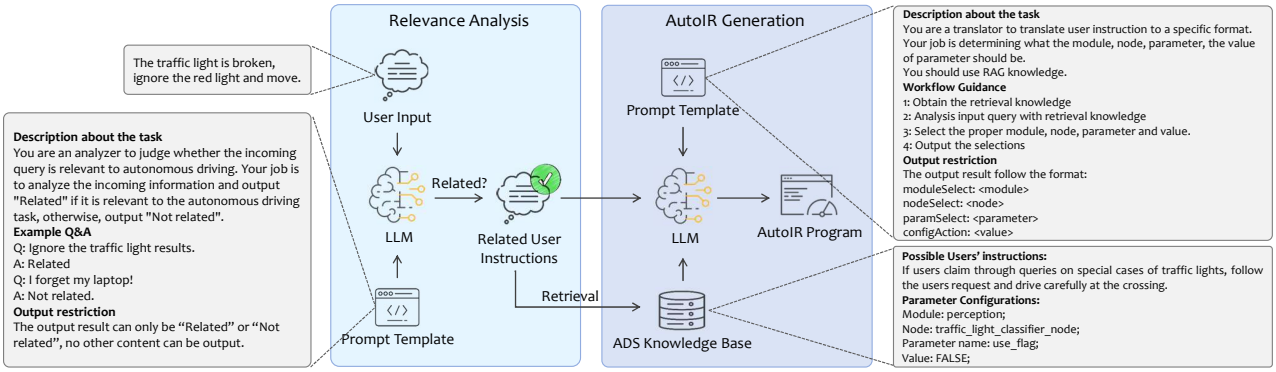


Fig. 3. The workflow of user instruction translation

(CoT) prompting [36], a technique that improves accuracy by providing context-specific examples within the prompt. Specifically, we feed the LLM not just the user's input but also a carefully designed prompt template. This template includes descriptive information about the task and several Q&A examples illustrating how to identify autonomous driving-related instructions (The left side of Fig. 3 gives examples of user input and prompt template).

If the LLM determines that the input sentence qualifies as a user instruction, it forwards the instruction to the next step for further processing. This approach ensures a lightweight yet effective method for filtering user input without requiring extensive model customization.

B. AutoIR Generation

In the second step, we generate an AutoIR program to implement a user instruction. Before diving into the details, we briefly introduce AutoIR. AutoIR is a custom-designed language that standardizes the translation output into a format that is understandable by the ADS. Essentially, it maps user instructions into Autoware's software constructs. Since Autoware is built on the ROS 2 middleware, we will first provide an overview of the structure of Autoware and ROS 2. This foundation will help readers to understand the role and design of AutoIR within the system.

1) The architecture of Autoware and ROS 2:

Autoware is an open-source software framework specifically designed to address the complexities of autonomous driving systems [15]. It employs a modular architecture that integrates all critical components required for autonomous vehicle operation, including sensing, localization, perception, planning, and control (as illustrated in Fig. 4). The sensing module collects raw environmental data from various sensors, such as LiDAR, cameras, and radar. This data is processed by the localization module, which determines the precise position and orientation of the vehicle within its environment. The perception module interprets the sensor data to identify objects, detect obstacles, and understand the surrounding environment. Based on this information, the planning module develops driving strategies, routes, and trajectories tailored to the vehicle's goals and the

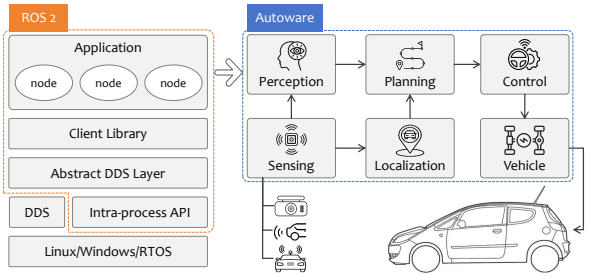


Fig. 4. The architectures of Autoware and ROS 2

environmental context. Finally, the control module executes these planned actions by managing the vehicle's actuators, such as steering, acceleration, and braking. These actions are further converted into low-level driver steps which control autonomous vehicles by the Vehicle module. These modules collaborate to enable efficient and accurate decision-making processes for autonomous vehicles.

ROS 2 (Robot Operating System 2) is an open-source framework for developing modular, scalable, and secure robotic software [22]. It is widely used in diverse applications, including autonomous driving, multi-robot systems, industrial automation, and healthcare robotics.

In ROS 2, an application is built based on nodes, which are lightweight, modular components that communicate using a publish-subscribe model (DDS in Fig. 4). Each Autoware module comprises a collection of ROS 2 nodes, each dedicated to specific tasks such as processing sensor data, detecting obstacles, or generating driving trajectories. This node-based design enables distributed processing and provides fine-grained control over the vehicle's functionalities. Key layers of ROS 2, including the Client Library, Abstract DDS Layer, DDS, and Intra-process API, offer the necessary abstractions for efficient inter-node communication and seamless data exchange. By leveraging ROS 2, Autoware ensures robust communication between nodes, enabling reliable handling of tasks ranging from sensor data ingestion to high-level decision-making and vehicle control.

AutoIR:
moduleSelect: planning
nodeSelect: behavior_path_planner
paramSelect: use_opposite_lane
configAction: FALSE

Fig. 5. An example of an AutoIR program

2) AutoIR Semantics and Generation:

The semantics of AutoIR define how user instructions are translated into metadata used for ROS 2 implementation. An AutoIR program consists of several domains of information, as exemplified in Fig. 5. The `moduleSelect` domain specifies the Autoware module that the user instruction will impact. For instance, if the user requests the ADS to change lanes, this instruction will affect the planning module. The `nodeSelect` domain identifies the specific node within the selected module that will be influenced, as each Autoware module may consist of multiple nodes. The `paramSelect` domain provides parameters for the selected node, guiding it to execute the desired actions. The `configAction` domain specifies the values to be assigned to these parameters, ensuring the node performs actions accordingly. The `Timer` domain specifies the lifetime of the user instruction.

To translate natural language user instructions into AutoIR, we leverage a Large Language Model (LLM). However, a major challenge lies in the fact that LLMs typically lack domain-specific knowledge of ADS and are unfamiliar with AutoIR semantics [10]. While one possible solution is to re-train the LLM with AutoIR examples and ADS domain knowledge, this approach is resource-intensive and requires a large amount of training data. Instead, we adopt the Retrieval-Augmented Generation (RAG) approach, which equips the LLM with an external knowledge base. RAG allows the LLM to retrieve relevant information during task execution, significantly reducing the need for re-training.

The effectiveness of RAG heavily depends on the quality of the knowledge base [11]. For example, one can use the entire Autoware manual as the knowledge base, but this is ineffective, as the manual contains a large amount of unrelated information, making it difficult for the LLM to locate the specific details it needs. To address this, we build a specialized ADS knowledge base derived from Autoware documentation. Each entry in the knowledge base pairs a driving scenario (representing a type of user instructions) with the corresponding AutoIR program. An example is illustrated in Fig. 3.

During AutoIR generation, the user instruction serves as input, triggering a retrieval query on the ADS knowledge base to extract relevant information. The retrieved information, along with the original user instruction, is then fed to the LLM to generate the corresponding AutoIR program. To further improve accuracy, we provide a structured prompt template as part of the input. This template guides the LLM on how to utilize the retrieved knowledge effectively during the generation process.

IV. Executing User Instructions

The execution of user instructions involves dynamically reconfiguring the ADS using the detailed information contained in AutoIR programs, enabling the ADS execution loop to carry out the specified driving actions.

A key challenge in this process is ensuring that user instructions do not result in unsafe driving behaviors. User instructions are often issued in special-case scenarios where the user's intent may conflict with the ADS's predefined rules. For instance, consider the malfunctioning traffic light scenario depicted in Fig. 1. In this situation, the user instructs the ADS to proceed through the intersection, which directly contradicts the ADS's predefined rule: "When a red light is observed at an intersection, keep the vehicle stationary." In these scenarios, following the user's instruction places the responsibility for safety on the user, as their command overrides the system's default behavior. Even though, the system should try to avoid executing instructions that are intentionally or carelessly unsafe. For instance, if the user instructs the vehicle to change lanes while cruising at high speed, such an instruction can be blocked to prevent potential accidents and preserve safety.

To address this, we develop a rule-based mechanism to validate and safeguard user instructions before integrating them into the ADS system. This mechanism ensures that user instructions meet predefined safety criteria and reduces the risks associated with unsafe commands.

A. Rule Base Design

We design the rules to safeguard user instruction execution offline using a simulation-based approach. This process begins by generating a set of AutoIR programs, which are manually validated for correctness. These programs are then tested in an ADS simulator to replicate typical driving scenarios. For each scenario, an AutoIR program is inserted, and the vehicle's status is observed through the ADS software. While the ADS provides numerous vehicle status parameters, we focus on the key parameters necessary for defining the rules:

- `Motion State`: indicates whether the vehicle is moving or stopped, along with the reasons for stopping.
- `Speed`: specifies the vehicle's current speed.
- `Perceptions`: provides information about the objects identified by the vehicle that may influence its driving decisions.

For example, to enforce a safety requirement such as "if the speed is above 70 km/h, do not change lanes", this condition is implemented in the simulated driving scenarios. By reading the key parameters of the vehicle's status during simulation, we derive rules that ensure safe instruction execution.

Each rule consists of two key components: the "Search Index" and "Conditions". The Search Index is used to identify and retrieve the relevant rule from the rule base based on the information provided by the AutoIR program. The Conditions specify safeguard parameters that ensure safe system behavior during the execution of user instructions.

An important condition is the `timer`, which serves as a critical safety mechanism. Since user instructions may override

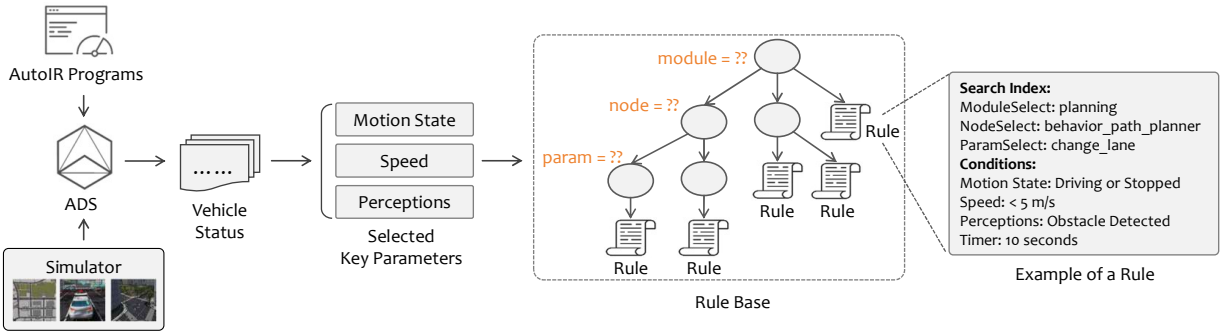


Fig. 6. The workflow for the design of the rule base

the ADS's default rules, the `timer` defines a specific duration, after which the system reverts to its default settings. This rollback mechanism ensures that deviations from standard behavior are temporary and safety is preserved. Currently, the timer values are manually and conservatively encoded in the rules during their design.

The rules generated through this process are organized into a tree structure to optimize searching. When an AutoIR program is received, the corresponding rule is located in the rule base using the search index, which facilitates efficient navigation and retrieval. Fig. 6 illustrates the workflow for generating the rule base, along with an example of a rule.

Upon receiving a user instruction in the form of an AutoIR program, execution proceeds only if the program matches a rule in the rule base and the specified conditions are satisfied. (The detailed validation process is introduced in the following subsection.) We note that the number of driving scenarios and AutoIR programs used during rule generation limits the number of generated rules and, consequently, the scope of acceptable user instructions. While it is impossible to enumerate all potential driving scenarios and user instructions due to their unlimited number, our current approach generates rules that reflect typical driving scenarios. But this ensures a safety baseline: unmatched user instructions and those that do not satisfy the conditions are ignored. System designers can, however, incrementally expand the rule base to accommodate more driving scenarios and support a broader range of user instructions over time.

B. Runtime Instruction Validation

At runtime, a dedicated software component is responsible for validating user instructions. The validation workflow is illustrated in Fig. 7. Importantly, whether a user instruction passes validation depends on the vehicle's status, which continuously changes during driving. As a result, upon the arrival of a user instruction, the validation process must repeatedly evaluate the instruction until its lifetime expires. Currently, the lifetime of a user instruction is manually set to 10 seconds. This setting can, of course, be further optimized based on the specific requirements of different instructions.

A user instruction represented as an AutoIR program is matched against the rules in the rule base, along with real-time

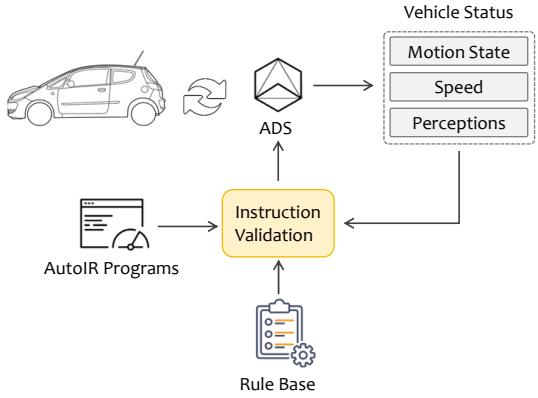


Fig. 7. The workflow of instruction validation

vehicle status data retrieved from the ADS. Only instructions that successfully pass validation, ensuring safety, are issued to the ADS for execution.

It is worth noting that each AutoIR program undergoes a final transformation into low-level ADS instructions before being fed to the ADS. This transformation is straightforward and does not require further elaboration here.

The instruction validation algorithm is presented in Algorithm 1. This algorithm takes user instructions in AutoIR format (I) and the rule base (R) as input and outputs whether the given user instruction should be executed (via an activation signal). Based on the information in the AutoIR program, Line 2 performs a search to find the corresponding rule in the rule base. Lines 3–8 describe the runtime checking process, during which the vehicle's status is continuously retrieved from the ADS to determine whether the conditions are met to execute the user instruction. This process terminates when the user instruction expires.

V. Experiments and Evaluation

In this section, we intend to evaluate (1) the accuracy and latency performance of instruction translation (Sec. V-A), (2) the effectiveness of instruction execution based on a simulation platform (Sec. V-B), and (3) the effectiveness of the overall

TABLE I
THE RESULTS OF ACCURACY EVALUATION FOR USER INSTRUCTION TRANSLATION

Methods	ModuleSelect Accuracy (%)	NodeSelect Accuracy (%)	ParamSelect Accuracy (%)	ConfigAction Accuracy (%)	Overall Accuracy (%)
Our Knowledge Base	95.5	95.5	93.5	87	87
Autoware Manual	81	64.5	50	32	32

Algorithm 1 Instruction Validation Algorithm (IVA)

Input: Input AutoIR I, Rule Base R
Output: Activation Signal

Function Instruction_Validation (I, R)
  $Rule \leftarrow Rule\_Searching(I, R)$
  while $(Instruct\ not\ expired)$ **do**
    $cur\_status \leftarrow Current\ vehicle\ status$
    if $Matching(cur\_status,\ Rule) = TRUE$ **then**
      return $Activated$
    end if
  end while
return $Not\_activated$
end Function

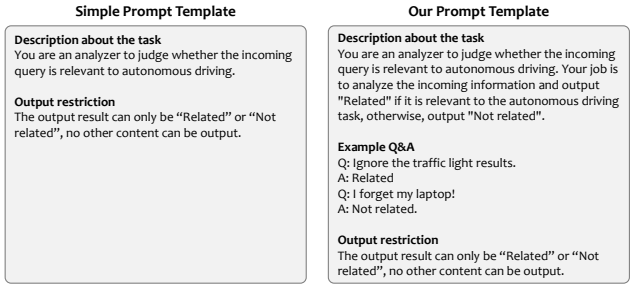


Fig. 8. Examples of the Simple Prompt template and our proposed prompt template for relevance analysis

Autoware.Flex system on a real-world autonomous vehicle (Sec. V-C).

A. Evaluation of Instruction Translation

1) The AutoIR Dataset: To evaluate the accuracy of user instruction translation, ground truths are essential. To this end, we develop a custom AutoIR dataset based on an in-depth analysis of Autoware to serve as the ground truth. This dataset is specifically created to address the absence of benchmarks in the existing literature for translating natural language into AutoIR. The dataset comprises pairs of user instructions in natural language and their corresponding AutoIR programs. These AutoIR programs are carefully crafted based on our extensive experience with Autoware and further verified through simulation to ensure they result in the correct driving behavior. The dataset includes 170 such pairs. Additionally, we incorporate 30 natural language user instructions unrelated to autonomous driving intended for testing purposes.

2) Accuracy Evaluation: We aim to evaluate not only the overall user instruction translation function but also its individual components, including relevance analysis, module selection, node selection, parameter selection, and configuration value assignment.

To assess the relevance analysis component, we focus on evaluating the effectiveness of our proposed prompt template, which leverages in-context learning. For comparison, we introduce a baseline template called `Simple Prompt`. The `Simple Prompt` template contains only a description of the relevance analysis task and output format constraints without providing Q&A examples. In contrast, our prompt template includes Q&A examples to guide the LLM more effectively. Examples of both templates are shown in Fig. 8.

To evaluate the other components and the overall user instruction translation function, the knowledge base that assists the LLM plays a critical role. In these experiments, we compare our specialized ADS knowledge base with a baseline knowledge base directly using the Autoware user manual. For consistency, our proposed prompt template is used during the relevance analysis step in all evaluations.

For all components and the overall translation function, accuracy is defined as the ratio of correctly processed results to the total number of test cases (200 items from the AutoIR dataset).

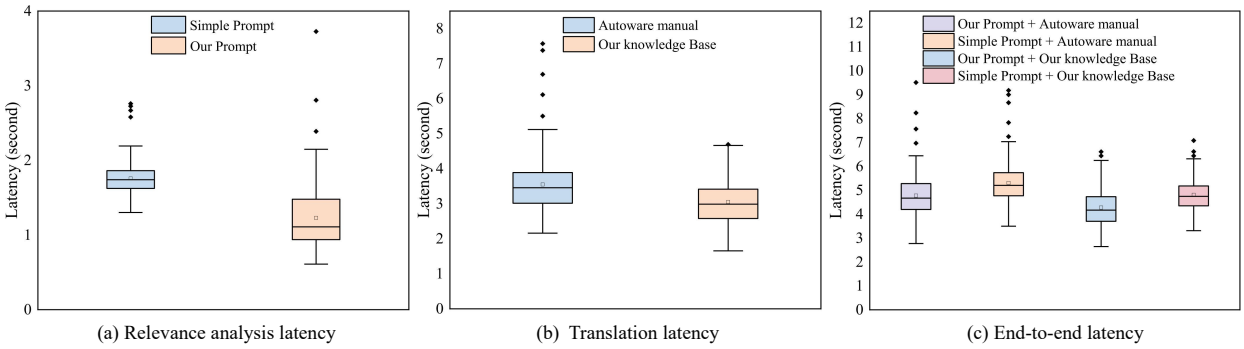
In terms of relevance analysis accuracy, our approach achieves 99%, significantly outperforming the `Simple Prompt`, which achieves only 92%. This demonstrates that the in-context learning prompting approach is more effective in guiding the LLM to correctly determine whether a user instruction in natural language is relevant.

The accuracy results for other components are summarized in Table I. Our specialized knowledge base shows a substantial improvement in accuracy compared to the approach using the Autoware manual as the knowledge base. This improvement is observed not only in the overall accuracy but also across all individual components. Notably, the ConfigAction step has the most significant impact on the overall accuracy of user instruction translation. This finding provides valuable insights for future optimization of the knowledge base, emphasizing the importance of enhancing the accuracy of the ConfigAction step.

3) Latency Evaluation: We evaluate the execution time overhead for three components: the relevance analysis step, the translation step, and the end-to-end user instruction translation process (including the latency of the first two steps).

For relevance analysis, we compare the latency of our approach with the `Simple Prompt` approach. For the translation step, we compare our approach with an alternative approach that uses the Autoware manual as the knowledge base in the RAG framework. To evaluate end-to-end latency, we consider four configurations combining the approaches used in the relevance analysis and translation steps.

We measure the time spent on each evaluation target (in units of seconds). The latency results were obtained by execut-



(a) Relevance analysis latency (b) Translation latency (c) End-to-end latency

Fig. 9. The results of latency evaluation for user instruction translation

ing the systems on a desktop computer equipped with an Intel Core i7-10700 CPU running at 2.90 GHz. The experimental results for latency evaluation are presented in Fig. 9. Fig. 9 (a) shows the latency results for the relevance analysis step. Our prompt template outperforms the compared approach in both average latency (1.5 s vs. 1.8 s) and peak latency (2.0 s vs. 2.5 s). This demonstrates that our prompt template not only improves accuracy (as evaluated previously) but also reduces the time overhead on the LLM to determine relevance. Further analysis reveals that latency variations and outlier values are primarily due to the QWenVL-Max Web API, which is susceptible to network fluctuations.

Fig. 9 (b) compares the latency of using different knowledge bases in the translation step. Once again, our approach outperforms the alternative approach that uses the Autoware manual as the knowledge base. This suggests that our specialized ADS knowledge base not only enhances accuracy but also reduces the computational burden on the LLM during the RAG process, thanks to the significantly smaller size of our ADS knowledge base.

Fig. 9 (c) illustrates the end-to-end latency results for the four configurations. This latency includes not only the time spent on the two individual steps but also the additional delays introduced by data communication between them. Our system, which employs the in-context learning prompt template for relevance analysis and utilizes our ADS knowledge base in the RAG framework, achieves the best performance, demonstrating the lowest average and peak latency.

The observed latency values are within an acceptable range, considering normal time delays in the autonomous driving loop. In emergency situations, such as a sudden lane change to avoid an unexpected obstacle, existing ADS systems should perform better than human intervention and should handle such cases autonomously without user involvement.

B. Evaluation of Instruction Execution

To evaluate the effectiveness of Autoware.Flex in executing user driving instructions, we employ the state-of-the-art Autoware simulator, AWSIM [17]. To simulate complex driving environments, we modified the Unity3D project of AWSIM.

The default AWSIM environment is based on the Shinjuku district in Japan and features a highly detailed 3D point cloud environment with semantic maps. Fig. 11 illustrates the simulated Shinjuku district environment and the Unity3D project setup used for AWSIM.

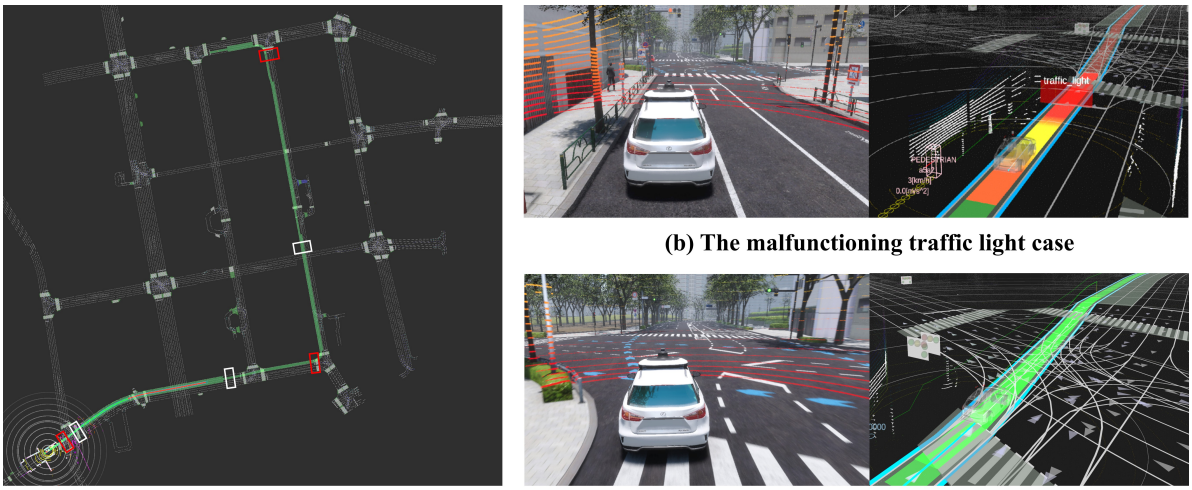
We design two scenarios within the simulator to validate whether our system can effectively and safely execute user instructions:

Malfunctioning traffic light: In this scenario, the traffic lights at an intersection malfunction, continuously displaying red lights. A traffic officer is assumed to be directing vehicles (though, due to the simulator's limitations, the officer is not visually represented in the scene). The user, understanding the situation, issues an instruction for the vehicle to ignore the traffic light and proceed through the intersection. Figure 10 (b) illustrates this scenario.

Restricted lane cruising: In this scenario, the user is searching for a destination building along the roadside. To facilitate the search, the user instructs the vehicle to cruise exclusively in the outermost lane (note that the map in the simulator is in Japan; the outermost lane is the left-most lane), making it easier to locate the destination. Figure 10 (c) depicts this scenario.

These two scenarios represent complex driving scenarios and user driving preferences respectively, which are representative. For each scenario, we use three different sentences to express the corresponding user instruction. For comparison, we also test native Autoware under the same conditions (without user instructions) to observe how it handled these scenarios. The simulation results are summarized in Table II.

In the malfunctioning traffic light scenario, three distinct user instructions are issued with the same intent to instruct the vehicle to proceed. Autoware.Flex successfully interpret and executed all three instructions. The simulation results show that the vehicle correctly move through the intersection by following the user's instructions. In contrast, native Autoware adhere strictly to its predefined rule: "I must wait until the traffic light turns green", and remain stationary. Table II also lists the generated AutoIR programs and the corresponding rules matched for this scenario.



(a) The driving route in the simulation

(b) The malfunctioning traffic light case

(c) The restricted lane cruising case

Fig. 10. Two scenarios to evaluate user instruction execution. (a) is the HDMap used in AWSIM, i.e., the routes taken by the vehicle. (b) shows the malfunctioning traffic light case; (c) shows the restricted cruising lane case.

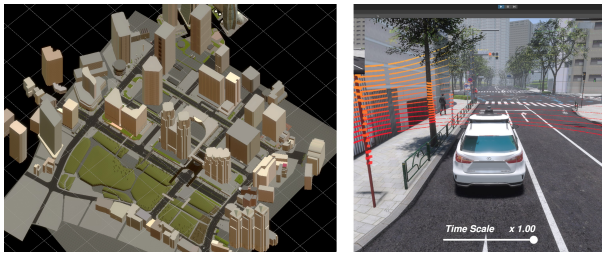


Fig. 11. The simulated driving environment in the AWSIM simulator. The left-side picture is an overview of the simulated Shinjuku district in Unity3D; the right-side picture shows the overview of the unity project.

Fig. 12. The prototype autonomous vehicle used in real-world evaluation

In the restricted lane cruising scenario, we again issue three different user inputs, all instructing the vehicle to remain in the outermost lane. The simulation results confirm that the vehicle stay in the outermost lane as directed. In comparison, native Autoware is unable to maintain the outermost lane and followed its own lane-selection rules, such as changing lanes based on traffic conditions. This user instruction is also associated with a timer specifying its validity duration. After the timer expired, the ADS revert to its predefined lane-control rules.

We also measure the time taken by Autoware.Flex for rule matching for each user instruction. Across all six experiments (three user instructions for each of the two scenarios), the maximum observed rule-matching delay is 0.77 ms for one round. This tiny delay attributes to the compact size of the rule base. At this level, the rule-matching delay can be considered negligible compared to the delays in other components within a single control cycle of the ADS.

C. Evaluation on a Real-world Autonomous Vehicle

To further validate Autoware.Flex in real-world environments, we develop a prototype autonomous vehicle, as shown in Fig. 12. The prototype vehicle is built on a drive-by-wire chassis and equipped with various sensors. The on-board computer used to run the ADS features an Intel Core i9-9900K CPU clocked at 3.6 GHz, with Autoware.Flex deployed on this system.

We conduct multiple tests using this prototype in a real-world parking lot. Fig. 13 provides drone-captured bird's-eye views of the vehicle in various experimental scenarios.

Experiment 1: Adjusting Distance to a Pedestrian

In this experiment, the vehicle encounter a pedestrian while driving. By default, the original ADS rule require the vehicle to stop one meter away from the pedestrian. However, if the user want to adopt a more conservative approach, they can issue an instruction to increase the stopping distance. Using user instructions, we direct the vehicle to stop farther from the pedestrian. As shown in Fig. 13 (a), the vehicle successfully

TABLE II
THE SIMULATION RESULTS FOR THE EVALUATION OF INSTRUCTION EXECUTION

Scenario	Case	User instruction	Translated AutoIR	Corresponding rule	Can Autoware.Flex handle?	Can Autoware handle?
Malfunctioning traffic light	1	The traffic light seems broken, ignore it.	moduleSelect: perception nodeSelect: traffic_light _classifier_node paramSelect: use_flag configAction: FALSE	Motion State: Stopped Speed: = 0 m/s Perceptions: Traffic Light Detected	Yes	No
	2	Do not follow the traffic light.				
	3	Traffic light is crazy! It is always red.				
Restricted lane cruising	1	I want you drive on the leftmost lane.	moduleSelect: planning nodeSelect: mission_planner paramSelect: lane_prefer configAction: LEFT	Motion State: Driving Speed: <5 m/s Perceptions: No Obstacle Detected	Yes	No
	2	Try to change to the leftmost lane.				
	3	I wanted to get as close to the left road as possible.				

stop approximately three meters away from the pedestrian, demonstrating the effectiveness of the user instruction.

Experiment 2: Circumventing a Traffic Cone

In this experiment, the vehicle encounter a traffic cone obstructing its lane. With only one lane available in each direction, the original ADS rule causes the vehicle to stop and remain stationary. However, the user, confident that the opposite lane is clear, issue an instruction for the vehicle to use the opposite lane to bypass the cone. Fig.13 (b) shows the vehicle stop in front of the cone before the instruction is issued. After the instruction is executed, the vehicle successfully move into the opposite lane to circumvent the cone, as shown in Fig.13 (c).

Experiment 3: Extended Stopping Time

In this experiment, the vehicle again encounter a traffic cone as an obstacle. According to the original ADS rule, the vehicle will briefly stop and then seek an alternate route to bypass the obstacle. However, the user issues an instruction to extend the stopping time in front of the obstacle. This experiment involve dynamic actions that can not be effectively represented with static images, so no photos are included for this scenario.

These real-world experiments strongly demonstrate Autoware.Flex's ability to correctly interpret and execute user driving instructions, even in complex and customized scenarios. For each experiment, the corresponding AutoIR programs and matched rules are provided in Table III.

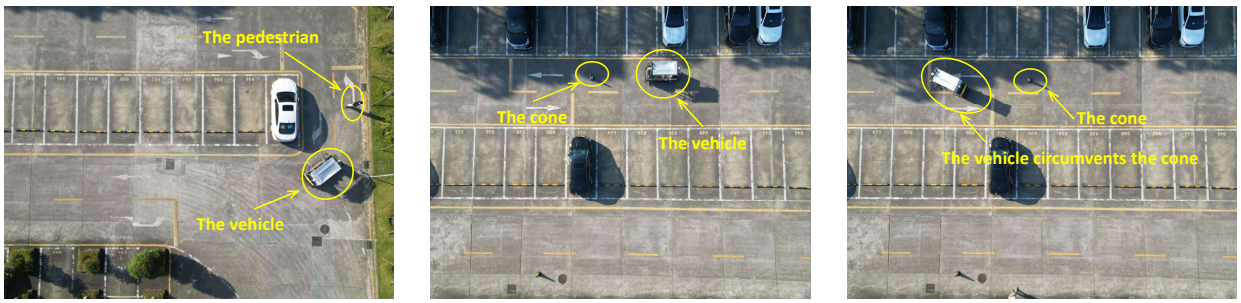
VI. RELATED WORK

The integration of Large Language Models (LLMs) into Autonomous Driving Systems (ADS) has attracted significant attention in recent research. Some studies explore the use of LLMs for trajectory planning in ADS. For instance, [5] proposes an object-level multimodal LLM architecture to enhance situational understanding in driving scenarios, while [24] demonstrates a method to adapt OpenAI GPT-3.5 models into reliable motion planners for autonomous vehicles. Similarly, [29] leverages the common-sense reasoning capabilities of LLMs like GPT-4 and Llama2 to improve vehicle planning. In [13], the potential of LLMs to interpret driving environments in a human-like manner is explored, highlighting their reasoning, interpretation, and memory capabilities in complex scenarios. Furthermore, [32] introduces a framework that utilizes LLMs to enhance decision-making processes in autonomous vehicles. Other research focuses on employing LLMs as intermediaries for human-computer interaction. For example, [35] presents a framework that integrates LLMs as a "co-pilot" for vehicles, aiming to facilitate interaction between humans and ADS. However, this approach involves direct interaction with the control model, which may introduce safety risks. Additionally, several works have explored implementing end-to-end ADS systems driven by LLMs, such as those described in [37], [28], and [9].

Despite their potential, LLM-driven ADS face significant challenges. As noted in [7] and [26], these systems often function as black boxes, making their decision-making processes opaque to humans. This lack of interpretability introduces considerable ethical and legal concerns. In contrast, rule-based modular ADS continue to excel in terms of safety and reliability. As discussed in [38], traditional ADS rely on well-defined rules and algorithms, ensuring predictable behavior across diverse conditions. These systems undergo rigorous testing and validation, providing consistent performance in various environments [4] [39]. Moreover, they incorporate multiple layers of redundancy and fail-safes to handle unexpected situations effectively [3] [15].

In this work, we leverage LLMs in a fundamentally different way. Rather than relying on LLMs for end-to-end decision-making, our primary contribution lies in integrating users' driving instructions into traditional rule-based modular ADS. This integration enables collaboration between humans and ADS during driving. Specifically, we use LLMs to translate users' driving instructions into ADS-domain-specific language. By adopting this approach, we effectively bridge the language gap between humans and ADS while retaining the modular structure of traditional ADS, thereby ensuring safety and reliability.

Considerable research has focused on developing domain-specific languages (DSLs) for the design, testing, and functional analysis of autonomous driving systems (ADS). For instance, [1] proposes CommonRoad, a composable road motion planning benchmark tailored for ADS design and testing. Similarly, [12] introduces Scenic, a probabilistic programming language aimed at the design and analysis of perception systems, particularly those based on machine learning. Other notable works include [27], which develops a DSL for capturing test scenarios that reflect the complexities



(a) **(b)** **(c)**

Fig. 13. Scenes from the real-world experiments: (a) depicts Experiment 1, Adjusting Distance to a Pedestrian, while (b) and (c) illustrate Experiment 2, Circumventing a Traffic Cone.

TABLE III
THE REAL-WORLD EXPERIMENT RESULTS

Case	User instruction	Translated AutoIR	Corresponding rule	Is user's instruction executed?
Pedestrian	Keep a larger distance from him	moduleSelect: planning nodeSelect: behavior_velocity _planner_node paramSelect: stop_margin configAction: 3.0	Motion State: Driving Speed: <5 m/s Perceptions: Obstacle Detected	Yes
Traffic Cone	Use the opposite lane to avoid it.	moduleSelect: planning nodeSelect: behavior_path_planner paramSelect: use_opposite_lane configAction: TRUE	Motion State: Driving Speed: <5 m/s Perceptions: Obstacle Detected	Yes
Waiting Time	Stop for a longer time	moduleSelect: planning nodeSelect: behavior_velocity _planner_node paramSelect: stop_duration configAction: 5.0	Motion State: Stopped Speed: = 0 m/s Perceptions: Obstacle Detected	Yes

of real-world road traffic conditions, and [31], which presents Lawbreaker, an automated framework for testing ADS compliance with real-world traffic regulations. Additionally, [6] designs a DSL specifically for describing traffic rules, while [21] proposes a DSL for aligning real-world accident reports in natural language with violation scenarios for ADS simulation testing. Further, [34] introduces μdrive, a DSL designed to give users direct control over ADS. By incorporating driver preferences, μdrive aims to enable safer, more stable, and more comfortable driving experiences.

While existing research emphasizes DSL design, this paper focuses on a different aspect of the problem. Although we propose AutoIR as a DSL for facilitating ADS operations, we acknowledge that other DSLs in the literature, such as μdrive, could potentially serve similar purposes. The primary challenge is not the design of the DSL itself but the faithful translation of user instructions from natural language into the DSL format. This translation process is critical to bridging the gap between human intent and ADS functionality.

VII. CONCLUSION

Existing autonomous driving systems (ADS) independently make driving decisions based on their perception of the environment. However, these systems face significant limitations: they cannot well handle complex scenarios where environmental understanding is inadequate and are unable to incorporate human driving preferences into their decision-making processes. This paper introduces Autoware.Flex, a system that enables users to provide instructions to the ADS, guiding it toward more appropriate driving decisions. Our system addresses two key challenges: translating natural-language human instructions into an ADS-compatible format and ensuring the safe execution of these instructions within the ADS framework. Experimental results from both simulators and a real-world autonomous vehicle demonstrate the effectiveness of the proposed approach.

The main contribution of this work is the novel approach to integrate human instructions into rule-based modular ADS. In the future, we aim to enhance the proposed system in the following ways: (1) developing more sophisticated AutoIR representations and leveraging advanced LLM techniques to handle complex natural language instructions; (2) enabling the system to infer the lifetime of a user instruction directly from natural language input; and (3) designing methods to automatically and incrementally expand and refine the knowledge base and rule base to support a broader range of scenarios.

ACKNOWLEDGMENT

This work was supported by the Research Grants Council of Hong Kong under Grant GRF 11208522. Figure 1 is generated by a generative AI tool - Doubao AI. Web link: http://www.doubao.com.

REFERENCES

[1] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 719–726.

[2] D. Atherton, "Incident 434: Sudden braking by tesla allegedly on self-driving mode caused multi-car pileup in tunnel," *AI Incident Database, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February*, vol. 13, p. 2023, 2022.

[3] Baidu, "Apollo: An open autonomous driving platform," [Online], 2019. [Online]. Available: https://github.com/ApolloAuto/apollo

[4] A. Carballo, D. Wong, Y. Ninomiya, S. Kato, and K. Takeda, "Training engineers in autonomous driving technologies using autoware," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 3347–3354.

[5] 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 *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 093–14 100.

[6] A. Collin, A. Bilka, S. Pendleton, and R. D. Tebbens, "Safety of the intended driving behavior using rulebooks," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 136–143.

[7] Y. Cui, S. Huang, J. Zhong, Z. Liu, Y. Wang, C. Sun, B. Li, X. Wang, and A. Khajepour, "Drivellm: Charting the path toward full autonomous driving with large language models," *IEEE Transactions on Intelligent Vehicles*, 2023.

[8] T. Deruyttere, S. Vandenhende, D. Grujicic, L. Van Gool, and M.-F. Moens, "Talk2car: Taking control of your self-driving car," *arXiv preprint arXiv:1909.10838*, 2019.

[9] Z. Dong, Y. Zhu, Y. Li, K. Mahon, and Y. Sun, "Generalizing end-to-end autonomous driving in real-world environments using zero-shot llms," *arXiv preprint arXiv:2411.14256*, 2024.

[10] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A survey on rag meeting llms: Towards retrieval-augmented large language models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6491–6501.

[11] M. Fatehkia, J. K. Lucas, and S. Chawla, "T-rag: lessons from the llm trenches," *arXiv preprint arXiv:2402.07483*, 2024.

[12] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, 2019, pp. 63–78.

[13] D. Fu, X. Li, L. Wen, M. Dou, P. Cai, B. Shi, and Y. Qiao, "Drive like a human: Rethinking autonomous driving with large language models," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 910–919.

[14] C. Hewitt, I. Politis, T. Amanatidis, and A. Sarkar, "Assessing public perception of self-driving cars: The autonomous vehicle acceptance model," in *Proceedings of the 24th international conference on intelligent user interfaces*, 2019, pp. 518–527.

[15] T. IV, "Autoware: The world's leading open-source software project for autonomous driving," 2022. [Online]. Available: https://autoware.org/autoware-overview/

[16] ——, "Autoware.universe documentation," 20222. [Online]. Available: https://autowarefoundation.github.io/autoware.universe/main

[17] ——, "Awsim: End-to-end digital twin simulation platform," 2023. [Online]. Available: https://autoware.org/awsim-end-to-end-digital-twin-simulation-platform

[18] N. Karanikolas, E. Manga, N. Samaridi, E. Tousidou, and M. Vassilakopoulos, "Large language models versus natural language understanding and generation," in *27th Pan-Hellenic Conference on Progress in Computing and Informatics*, 2023, pp. 278–290.

[19] S. Khemka, "Incident 347: Waymo self-driving taxi behaved unexpectedly, driving away from support crew," *AI Incident Database, Khoa Lam (Ed.). Responsible AI Collaborative. Retrieved February*, vol. 13, p. 2023, 2021.

[20] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[21] Y. Lu, Y. Tian, Y. Bi, B. Chen, and X. Peng, "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*, 2024, pp. 376–388.

[22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[23] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, vol. 1, 2020.

[24] J. Mao, Y. Qian, and et al., "Gpt-driver: Learning to drive with gpt," *arXiv preprint arXiv:2310.01415*, 2023.

[25] S. McGregor, "Incident 4: Uber av killed pedestrian in arizona," *AI Incident Database, Sean McGregor (Ed.). Responsible AI Collaborative. Retrieved February*, vol. 13, p. 2023, 2018.

[26] C. Pan, B. Yaman, T. Nesti, A. Mallik, A. G. Allievi, S. Velipasalar, and L. Ren, "Vlp: Vision language planning for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[27] R. Queiroz, T. Berger, and K. Czarnecki, "Geoscenario: An open dsl for autonomous driving scenario representation," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 287–294.

[28] H. Shao, Y. Hu, L. Wang, G. Song, S. L. Waslander, Y. Liu, and H. Li, "Lmdrive: Closed-loop end-to-end driving with large language models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 120–15 130.

[29] S. Sharan, F. Pittaluga, M. Chandraker *et al.*, "Llm-assist: Enhancing closed-loop planning with language-based reasoning," *arXiv preprint arXiv:2401.00125*, 2023.

[30] J. Shen, N. Tenenholtz, J. B. Hall, D. Alvarez-Melis, and N. Fusi, "Tag-llm: Repurposing general-purpose llms for specialized domains," *arXiv preprint arXiv:2402.05140*, 2024.

[31] Y. Sun, C. M. Poskitt, J. Sun, Y. Chen, and Z. Yang, "Lawbreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.

[32] Y. Sun, C. M. Poskitt, X. Zhang, and J. Sun, "Redriver: Runtime enforcement for autonomous vehicles," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[33] B. Wang, Z. Wang, X. Wang, Y. Cao, R. A Saurous, and Y. Kim, "Grammar prompting for domain-specific language generation with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[34] K. Wang, C. M. Poskitt, Y. Sun, J. Sun, J. Wang, P. Cheng, and J. Chen, "mu drive: User-controlled autonomous driving," *arXiv preprint arXiv:2407.13201*, 2024.

[35] S. Wang, Y. Zhu, Z. Li, Y. Wang, L. Li, and Z. He, "Chatgpt as your vehicle co-pilot: An initial attempt," *IEEE Transactions on Intelligent Vehicles*, 2023.

[36] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[37] Z. Xu, Y. Zhang, E. Xie, Z. Zhao, Y. Guo, K.-Y. K. Wong, Z. Li, and H. Zhao, "Drivegpt4: Interpretable end-to-end autonomous driving via large language model," *IEEE Robotics and Automation Letters*, 2024.

[38] Z. Yang, X. Jia, H. Li, and J. Yan, "Llm4drive: A survey of large language models for autonomous driving," in *NeurIPS 2024 Workshop on Open-World Agents*, 2023.

[39] Z. Zang, R. Tumu, J. Betz, H. Zheng, and R. Mangharam, "Winning the 3rd japan automotive ai challenge-autonomous racing with the autoware. auto open source software stack," in *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2022, pp. 1757–1764.