This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2026.3703032

Exploring Critical Testing Scenarios for Decision-Making Policies: An LLM Approach

Weichao Xu, Huaxin Pei, Jingxuan Yang, Yuchen Shi, Yi Zhang, *Senior Member, IEEE* and Qianchuan Zhao, *Senior Member, IEEE*

***Abstract*—Recent advances in decision-making policies have led to notable progress in various fields such as autonomous driving and robotics. However, testing these policies remains crucial with the existence of critical scenarios that may threaten their reliability. Despite ongoing research, challenges such as low testing efficiency and limited scenario diversity persist, primarily due to the complexity of the policies and their operating environments. To address these challenges, this paper proposes an adaptable Large Language Model (LLM)-driven adaptive testing framework to explore critical and diverse testing scenarios for decision-making policies. Specifically, we introduce a "generate-test-feedback" pipeline, leveraging templated prompt engineering to harness the world knowledge and reasoning abilities of LLMs. Additionally, a multi-scale scenario generation strategy is proposed to overcome the limitations of LLMs in making fine-grained adjustments, thereby further enhancing testing efficiency. Furthermore, during the feedback phase, a well-designed scenario evaluation module assesses the criticality and diversity of the generated scenarios for adaptive testing. The proposed LLM-driven method is extensively evaluated on five widely recognized benchmarks, demonstrating that our proposed method significantly outperforms baseline methods in uncovering both critical and diverse scenarios. These findings highlight the great promise of LLM-driven schemes in advancing the testing of decision-making policies.**

***Index Terms*—Large language models (LLMs), testing scenario generation, critical testing scenario, decision-making policies.**

I. Introduction

In recent years, the policies solving sequential decision-making problems have achieved promising results across various fields such as autonomous driving [1]–[4], robotics [5]–[7], and Go [8]. With advancements in artificial intelligence, neural network-based decision-making policies are now capable of matching or even surpassing human performance. Despite their remarkable effectiveness, these policies face challenges related to interpretability, robustness, and reliability, leading to critical scenarios that may trigger failures during application. To address these challenges, it is essential to develop efficient and comprehensive testing methods for evaluating decision-making policies.

Testing decision-making policies involves uncovering a series of states that represent critical scenarios, which expose incorrect behaviors in the target policy and cause it to fail at its task. However, since these policies are predominantly developed using neural networks, their "black-box" nature poses challenges to understanding and predicting their behaviors. The continuous interactions between the policy and its environment also bring difficulty in predicting critical scenarios. Additionally, the potentially infinite number of states and the high-dimensional state space involved can complicate testing, further hindering the search for critical scenarios.

Increasing attention has been given to developing testing methods to address the challenges outlined above. However, existing methods [9]–[14] often rely heavily on complex designs and prior knowledge, which limits their adaptability to large-scale and diverse tasks. Moreover, the flexibility of these frameworks is constrained by their specific design elements. While simplifying designs to improve generalizability can enhance adaptability, it may undermine the effectiveness of the testing methods, making it difficult to achieve both efficiency and universality simultaneously. Therefore, an efficient adaptive testing method is required—one that can dynamically generate and adjust testing scenarios based on the real-time performance of the target policies. This work aims to explore a practical solution to this challenge.

The pervasive emergence of Large Language Models (LLMs) has recently introduced innovative approaches to tackling complex problems across diverse fields [15]–[17]. By leveraging extensive training data, LLMs demonstrate a remarkable level of intelligence, encompassing broad world knowledge and advanced reasoning capabilities. In the context of testing decision-making policies, LLMs could, in theory, apply their inherent common sense and reasoning skills to analyze challenging scenarios in diverse environments. This presents the possibility for adaptive scenario generation with minimal human involvement. Additionally, techniques like prompt engineering [18] suggest that LLMs could iteratively learn from past testing experiences, enabling more efficient adaptive testing. Their creative capabilities also show promise for generating novel and diverse scenarios, even from limited initial inputs or reference cases. By leveraging this creativity, LLMs could modify and adjust scenarios at multiple scales, facilitating cross-regional exploration and uncovering previously uncharted areas.

Built upon the aforementioned strengths of LLMs, this work

This work was supported in part by the National Natural Science Foundation of China under Grant 62503259. (*Corresponding author: Huaxin Pei.*)

Weichao Xu, Huaxin Pei, Jingxuan Yang, and Yuchen Shi are with the Department of Automation, Tsinghua University, Beijing 100084, China (email: xwc23@mails.tsinghua.edu.cn, phx17@tsinghua.org.cn, yangjx20@mails.tsinghua.edu.cn, shiyuche21@mails.tsinghua.edu.cn).

Yi Zhang is with the Department of Automation, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China, and also with the Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, Nanjing 210096, China (e-mail: zhyi@mail.tsinghua.edu.cn).

Qianchuan Zhao is with the Center for Intelligent and Networked Systems, Department of Automation, Tsinghua University, Beijing 100084, China (e-mail:zhaoqc@tsinghua.edu.cn).

investigates how their reasoning and learning capabilities can be harnessed to generate critical testing scenarios, aiming at efficient and versatile testing of decision-making policies. However, several key challenges emerge when developing an LLM-driven approach: *i)* In the absence of well-defined reference points, LLM-generated responses often lack depth and complexity, making it difficult to directly create intricate scenarios. *ii)* Inherent limitations, such as hallucinations, hinder the ability to guide LLMs to focus on the target environment, reason accurately, and evolve based on human input or historical experience. *iii)* The probabilistic nature of LLM text generation results in responses biased toward common patterns, leading to outcomes that may lack precision or nuance. Prior work has shown that LLMs have limited ability to instantiate precise parameter values from specified ranges [19]. We observed the same phenomenon in our experiments: within a defined range (e.g., [0,100]), precise values such as 51.7 are considerably more difficult for LLMs to generate than rounded values such as 50. A detailed example is provided in Section V-C.

To cope with those challenges above, this paper introduces LLMTester, an innovative LLM-driven adaptive testing framework designed to evaluate decision-making policies. As illustrated in Fig. 1, the proposed framework consists of four key modules, i.e., a scenario database, a scenario generator powered by LLMs, a scenario testing module, and a generalized scenario evaluation module. With the proposed framework, a dynamic "generate-test-feedback" pipeline is formulated, enabling flexible and adaptive testing guided by diverse inputs. More specifically, the details and core concepts underlying LLMTester are elaborated as follows:

Firstly, we design a scenario database module that stores and maintains seed scenarios, which serve as reference points for the LLM-driven scenario generator. By grounding the scenario generator in these seed scenarios, the database provides valuable prior knowledge to the LLM, preventing the generation of random or trivial outputs and ensuring the consistent creation of challenging and meaningful scenarios.

Secondly, we propose a structured prompt pipeline that integrates feedback from historical testing data and expert experience. In addition, techniques such as Chain-of-Thought (COT) [20] are incorporated to guide LLMs toward deeper reasoning and analysis. Furthermore, based on the characteristics of decision-making tasks, we summarize several key elements that support the design of efficient prompts and facilitate the rapid transferability to new tasks.

Thirdly, it has been observed that LLMs face challenges in making fine-grained adjustments to near-failure scenarios. In response, we evaluate the potential of seed scenarios to evolve into critical scenarios through small random perturbations. Subsequently, for low-potential scenarios, the LLM-based generator is employed for large-scale searches, while small-scale random mutations are applied to high-potential scenarios to leverage their near-failure characteristics. This multi-scale generation strategy effectively identifies critical scenarios while minimizing resource consumption.

Lastly, scenarios after testing are evaluated for criticality and diversity using the scenario evaluation modules. The evaluation results provide feedback for online optimization, allowing the LLM to progressively adapt to the target policies. A key advantage of LLMTester over previous works is the flexibility of its evaluation mechanism. Leveraging the LLM's understanding capabilities, testers can specify various types of feedback and directly supply testing information to the LLM, without the need to explicitly define how this information should be utilized.

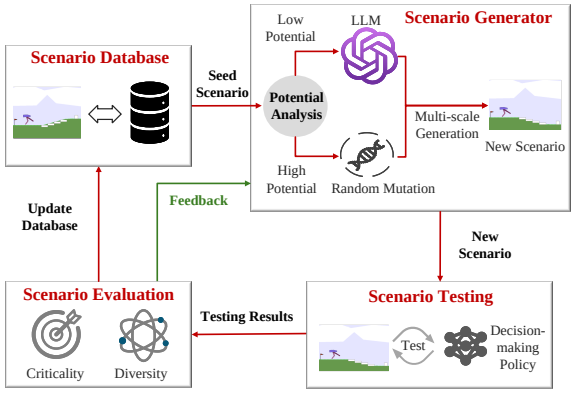


Fig. 1: Overview of LLM-driven adaptive testing framework.

The proposed LLMTester is evaluated on five widely recognized benchmarks. The results show that our method significantly outperforms the baseline in terms of the discovery rate of critical scenarios, while also ensuring the diversity of failure scenarios. Further ablation and comparative experiments underscore the effectiveness and robustness of our LLMTester.

Compared with previous works, LLMTester extends the application of LLMs to the exploration of critical testing scenarios for general decision-making policies, with a focus on adaptive testing that integrates expert experience and testing feedback. The contributions of this work can be summarized as follows:

i) We present a general LLM-driven adaptive testing framework with a "generate-test-feedback" pipeline that integrates both testing feedback and expert knowledge, enabling efficient adaptive testing of decision-making policies.
ii) A powerful prompt pipeline is designed to fully leverage the capabilities of LLMs in scenario generation. Based on an analysis of the characteristics of decision-making tasks, we extract and summarize key elements from this prompt to enable efficient and rapid testing of new policies.
iii) To address the limitations of LLMs in making fine-grained adjustments, a multi-scale generation strategy with adaptive potential analysis is proposed, which further boosts testing efficiency while reducing resource consumption.
iv) Comprehensive experimental results, derived from testing five policies across four distinct environments, demonstrate the effectiveness and general applicability of the proposed method.

The rest of this paper is organized as follows. Section II presents the preliminaries, including the key concepts and definitions used throughout this work. Section III reviews

the related work. In Section IV, we introduce the details of our method, including the adaptive testing framework, LLM-based scenario generator, and multi-scale generation strategy. Section V describes the experimental setup and evaluation results, while Section VI presents the discussions and threats to validity. Section VII briefly concludes the paper.

II. Preliminary

A. Decision-Making Policies

Sequential decision-making refers to problems in which an agent interacts with an environment in a step-wise manner to achieve long-term objectives. These problems are commonly modeled as a Markov Decision Process (MDP), represented as $\langle S, A, T, R \rangle$.

Here, S denotes the state space and A the action space. At each timestep t, the agent observes a state $s_t \in S$ and selects an action $a_t \in A$. The transition function T governs the environment dynamics, typically written as $s_{t+1} = T(s_t, a_t)$. The reward function R assigns an immediate scalar feedback $r_t = R(s_t, a_t)$, reflecting the quality of the selected action.

A solution to an MDP is a policy π, which maps states to actions and can be deterministic or stochastic (i.e., defining a distribution $\pi(a \mid s)$). The objective of most decision-making algorithms is to maximize the expected cumulative reward over a trajectory:

$$\mathbb{E}\left(\sum_{t=0}^{T} \gamma^t r_t\right), \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor. This cumulative reward serves as the primary quantitative measure of policy performance.

B. Testing Decision-Making Policies

In sequential decision-making, the observed state s represents the information perceived by an agent from the environment at a given timestep. To systematically evaluate the agent, it is necessary to configure the environment itself. The environment can be described by a set of parameters, which together form a scenario vector, also denoted as s. In this paper, unless otherwise specified, s refers to the scenario vector used for testing.

A straightforward method for generating test scenarios is random mutation, which perturbs the scenario vector by adding a random noise vector:

$$s' = s + \epsilon, \tag{2}$$

where ϵ is a random vector sampled from a suitable distribution. This approach provides a basic way to explore the environment and identify situations where the agent's policy may fail, while, as noted in related work (Section III), more advanced scenario generation methods also exist.

III. Related Work

A. Testing of Decision-making Policies

Testing decision-making policies involves generating test cases that trigger failures. The most straightforward approach is to directly modify the inputs to the decision-making policies. Various testing methods [21]–[26] focus on designing algorithms for finding special test inputs to evaluate the security and robustness of DNN-based policies. However, in some constrained environments, the casually generated inputs of neural networks may not correspond to real-world scenarios, making such inputs practically irrelevant. Furthermore, since internal policies are often highly encapsulated, many approaches [27]–[40] test decision-making policies through scenario generation.

1) Data-driven: Data-driven methods rely on pre-collected data or real-time interaction with the environment to implicitly acquire domain knowledge and enhance testing efficiency, such as reinforcement learning (RL)- or deep learning (DL)-based approaches. For instance, [29]–[32] employ methods based on RL and Monte Carlo tree search to perform adaptive stress testing on autonomous driving, thereby generating challenging test scenarios. For testing competitive game agents, AdvTest [33] designs and adds constraints to guide adversarial agent training, with the goal of exposing a more diverse range of failure scenarios. However, these methods often incur additional testing costs due to parameter tuning and real-time interaction. In parallel, methods leveraging extensive training data have also been continuously proposed. [34]–[36] construct new scenario datasets by analyzing extensive scenario data and sampling safety-critical scenarios, while NeuralNDE [37] utilizes a Transformer-based model to learn from real-world data and efficiently generate synthetic data with distributions that are closely similar to real-world distributions.

2) Knowledge-driven: Since data-driven methods require large amounts of data or extensive interaction with the environment to achieve satisfactory performance, many approaches [38]–[44] focus on manually extracting effective rules to rapidly improve testing efficiency or enhance generalizability. To identify unique traffic violations, Autofuzz [38] uses a neural network for seed selection and mutation, and implements a grammar-based fuzzing framework. BehAVExplor [39] employs diversity and violation feedback to help discover critical driving scenarios, incorporating an adaptive mechanism that applies different types of mutations based on seed energy. Meanwhile, scenoRITA [40] introduces evolutionary algorithms to search for critical scenarios. However, these methods rely on rules tailored to specific tasks, sacrificing generalizability across different tasks in favor of efficiency. Consequently, recent works [41]–[44] have emerged that focus on higher-level knowledge, designing universal testing frameworks based on the shared characteristics of decision-making policies. MDPFuzz [41] proposes the first general-purpose fuzz testing framework for models solving MDPs. Built upon MDPFuzz, SeqDivFuzz [42] enhances the testing efficiency by terminating non-diverse sequences early, based on sequence diversity inference. GMT [43] trains a generative diffusion model to generate diverse, failure-triggering test cases for decision-making policies and continuously fine-tunes the generative model during testing. For multi-agent systems, MASTest [44] calculates the criticality of states (considering both exploitation and exploration) and perturbs the action at critical states. However, the pursuit of generality in these

methods comes at a cost: *they do not account for the specific characteristics of the environment under test, which limits the efficiency of the testing process*.

Existing methods either depend on data or task-specific designs to improve testing performance for particular tasks, or they forgo domain knowledge in pursuit of more generalized testing frameworks. Balancing efficiency and generalizability remains a significant challenge. In this paper, we propose leveraging LLMs as a solution. Pretrained on extensive datasets, these models demonstrate powerful comprehension capabilities and extensive knowledge. LLMTester utilizes LLMs to perform environment-specific searches for critical scenarios, even in the absence of abundant data or manual design, thereby enabling both efficient and generalizable scenario generation.

B. Testing with LLMs

As LLMs have been applied across various fields, [45]–[49] is seeking to utilize the capabilities of LLMs to produce test cases for the purpose of testing a specific algorithm or software. For example, LIBRO [47] proposes using LLMs to reproduce a given bug by querying the model to generate test methods that align with the bug report. TitanFuzz [48] applies fuzzy testing to deep learning libraries, using LLMs' code generation and completion abilities to mutate seeds. As a result, it successfully identifies previously unknown bugs in TensorFlow/PyTorch. InputBlaster [49] first utilizes LLMs to generate valid text, and then applies mutation rules to generate unusual inputs that trigger application crashes.

Similarly, LLMs have been widely developed for scenario generation in areas such as autonomous driving. ChatScene [50] uses an LLM to generate descriptions of safety-critical scenarios and retrieves corresponding Scenic code snippets from a pre-constructed database. However, the design and acquisition of Scenic code remain challenging, which limits its applicability and generalization to other tasks. ChatSim [51] engages multiple LLM agents in collaborative efforts to understand user commands and generate corresponding photo-realistic scene videos, but it is designed for generating videos based on user requests, rather than for producing critical scenarios. SeGPT [52] leverages real-world scenarios, allowing an LLM to modify them and generate synthetic datasets for trajectory prediction testing, with a primary focus on modifying vehicle trajectories for datasets tailored to evaluating and training trajectory prediction algorithms. LLMScenario [53] extracts naturalistic risky trajectories from a database and guides the LLM to produce different challenging trajectories, including the ego vehicle's trajectory; however, LLMScenario is not a specialized testing framework, as it does not address challenges posed by the black-box nature of the target policies. OmniTester [54] takes a further step by employing multimodal LLMs to generate road networks and vehicle configurations with a carefully designed framework. However, it generates testing scenarios by constructing road layouts and positioning vehicles using tools like SUMO [55], which imposes strict constraints on scenario formats and environments. The methods described above lack an effective online optimization framework during scenario generation, limiting their ability to identify flaws in policies, especially in tasks where predicting the agent's behavior is more complex than in autonomous vehicle applications. Consequently, the scenarios generated by these methods are not always sufficiently critical.

To summarize, these existing LLM-based testing methods are primarily designed for specific domains, particularly in autonomous driving and software testing, and do not address generalized scenarios. This limitation results in a lack of testing capability for more universal systems. In contrast, our approach is aimed at general decision-making policies, outlining a scenario generation template and guiding LLM-driven scenario generation through an adaptive testing framework.

IV. Methodology

We propose LLMTester, a general LLM-driven adaptive testing framework, designed to efficiently test decision-making policies by exploring critical scenarios. In this section, the overall testing framework with the four proposed modules is first presented in Section IV-A. In Section IV-B, we focus on the LLM-based scenario generator within this framework. To address the inherent challenges faced by LLMs and further enhance the testing efficiency, a multi-scale generation strategy is proposed in Section IV-C.

A. The Testing Framework

Fig. 2 shows the overall procedure of LLMTester. The whole framework can be divided into four modules: *Scenario Database*, *Scenario Generator*, *Scenario Testing*, and *Scenario Evaluation*. Each of these modules is designed with the flexibility to adapt to different policies and environments.

1) Scenario Database: In our proposed framework, a scenario database is primarily established to store key parameters representing each scenario (e.g., initial states) along with associated testing information, both of which are fundamental to the testing process. For example, in the BipedalWalker environment provided in Fig. 2, where a robot needs to traverse different terrain, the initial states correspond to combinations of terrain parameters, while the testing information could include evaluations of the robot's performance in this scenario. Each scenario in the database serves as a seed for further modification by the scenario generator. During testing, seed scenarios are continuously selected and provided to the generator to create new scenarios, while the accompanying test information helps enhance the generator's capabilities. In this way, the generation and maintenance of the scenario database are refined through various guidance techniques to meet different testing objectives. For instance, the database can be updated to include only scenarios with higher criticality and greater diversity. This update process is driven by the performance of the policy under test, ensuring that the seed scenarios provide dynamic and relevant information for the policy.

Without loss of generality, the scenario database can be constructed through random sampling after parameterizing the elements within the environment, or by sampling from pre-existing datasets. For example, in autonomous driving, testing

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2026.3703032

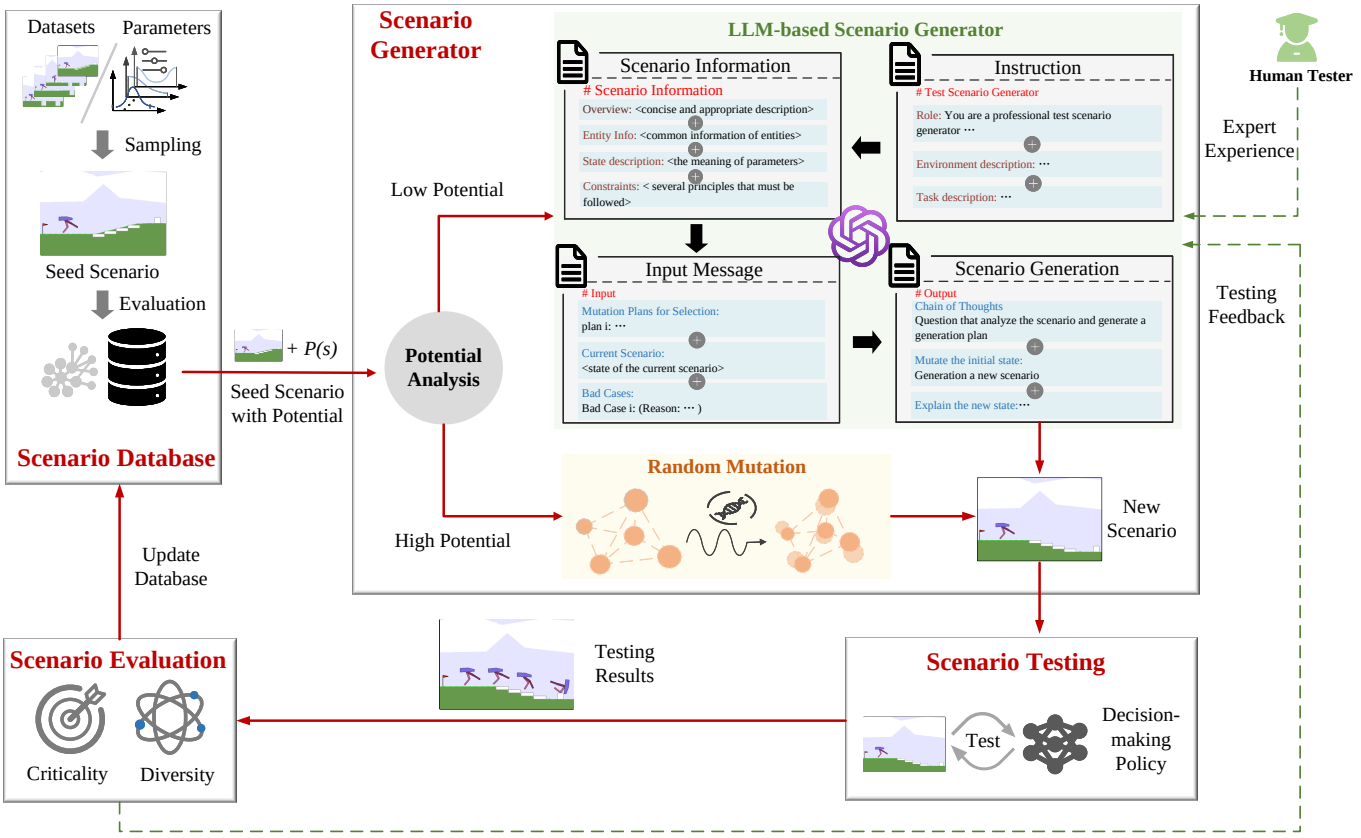


Fig. 2: Schematic workflow of our LLM-driven adaptive testing framework, namely LLMTester. *Scenario Database* provides the seed scenarios as references for generating new scenarios. *Scenario Generator*, through prompt engineering, harnesses the intelligence of the LLM to efficiently generate critical scenarios. To address the inherent limitations of LLMs, a multi-scale generation strategy is employed to analyze the potential of seed scenarios to escalate into critical scenarios, guiding the decision of whether to use the LLM-based generator. *Scenario Testing* tests the policy in the given scenarios. *Scenario Evaluation* assesses the criticality and diversity based on the testing results, providing feedback to the LLM for self-improvement.

scenarios can be initialized by randomly sampling the vehicle's location and yaw, or by extracting data from datasets such as KITTI [56].

2) Scenario Generator: The scenario generator, upon receiving seed scenarios sampled from the database, generates new testing scenarios based on the seeds. Simultaneously, it incorporates the testing feedback from the scenario evaluation module to refine the generation process. For example, as illustrated in Fig. 2, the scenario generator uses a seed scenario in the BipedalWalker environment to produce a new terrain configuration. The core component of this module is an LLM-based scenario generator. The LLM-based generator transforms seed scenarios, testing feedback, and expert knowledge into prompts using a standardized template, thereby guiding the LLM to generate new and more challenging scenarios. In this paper, the scenario generator is also capable of creating scenarios through random mutation. Specifically, a multi-scale generation strategy is implemented to combine the advantages of both the LLM-based generator and random mutation, enhancing the efficiency of exploring critical testing scenarios. The details of the scenario generator can be found in Section IV-B and IV-C.

3) Scenario Testing: The new scenarios generated by the scenario generator are integrated to configure the environment and conduct a test for the target policy. Subsequently, the actual performance of the policy under test is evaluated and utilized for adaptive testing.

4) Scenario Evaluation: The scenario evaluation module assesses both seed scenarios and the newly generated ones across two primary dimensions: criticality and diversity. In practice, evaluations are often based on trajectories collected during testing. For example, in the BipedalWalker environment, the evaluation can consider the robot's state at each timestamp, such as joint angles and positions. These evaluations guide three key processes: selecting seed scenarios, updating the scenario database, and generating new scenarios. The measurement of criticality and diversity utilizes specific metrics designed to align with the performance of the policies being tested. Based on the evaluation results, the module determines the importance of each scenario and decides whether it should be sampled or included in the database. Additionally, these testing data can be fed back to the scenario generator to support adaptive testing.

The evaluation module can be designed flexibly based on the target environment, as long as the evaluation scores are

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2026.3703032

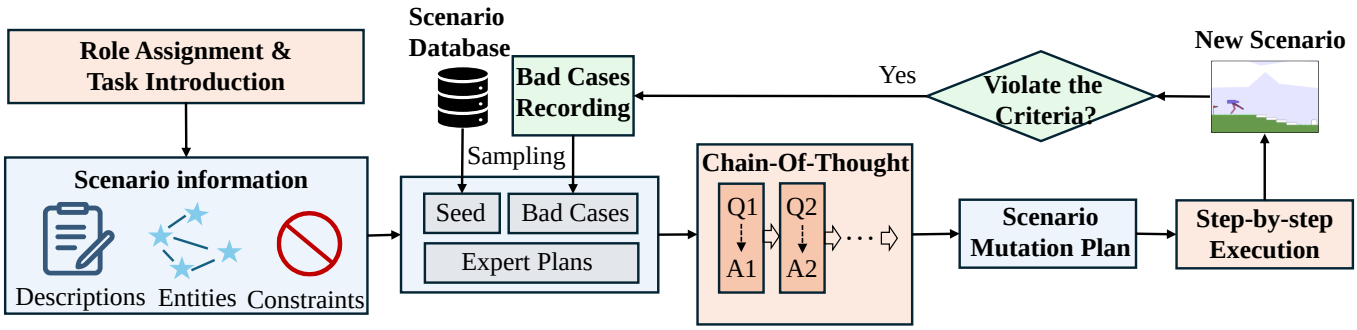


Fig. 3: The prompt pipeline for LLM-based scenario generation.

correlated with the importance of the scenarios. Here, we recommend the evaluation method proposed in MDPFuzz [41]. When selecting seed scenarios, sensitivity is computed as the sampling weight by randomly mutating the initial state of the seed scenario, and is calculated as:

$$\rho = \frac{|r_{\text{seed}} - r_{\delta}|}{||\Delta||_2}, \tag{3}$$

where r_{seed} and r_{δ} are the cumulative reward obtained by the decision-making policy under the seed scenario and mutated scenario, respectively, and Δ represents a random permutation. A seed scenario with higher sensitivity is more likely to exhibit significant changes in response to small perturbations, thereby increasing the likelihood of failure when new scenarios are generated from it. Specifically, seed selection is implemented via weighted sampling based on sensitivity, where higher-sensitivity seeds are assigned larger sampling probabilities, while all seeds retain non-zero selection probability. This design biases sampling toward critical scenarios while preserving stochastic exploration over the full scenario space.

When updating the database, freshness[41] and cumulative reward, as defined in MDPFuzz, are used to measure both the coverage of the explored scenario space and the criticality of seed scenarios. Specifically, a newly generated scenario replaces its original counterpart if it exhibits lower reward or reduced coverage. This replacement mechanism enables the seed pool to evolve over time while continuously maintaining scenario diversity. In addition, once a scenario triggers a failure, its corresponding seed is removed to prevent redundant exploitation and to encourage exploration of unexplored regions.

Overall, the seed selection and update mechanisms prioritize scenarios that are more likely to induce failures, while simultaneously maintaining or improving diversity in the seed pool. Based on such a seed pool, the scenario generator is encouraged to produce a larger number of critical and diverse scenarios. Furthermore, a specialized evaluation mechanism for adaptive testing, as introduced in Section IV-B, will be employed in scenario generation, along with a multi-scale generation strategy discussed in Section IV-C.

B. LLM-based Scenario Generator

The LLM-based scenario generator operates by using environment-specific prompts and integrating expert knowledge along with feedback from prior tests. It modifies seed scenarios to create new ones with a higher likelihood of inducing failures. The process is outlined as follows:

$$s_{\text{new}} = \mathcal{G}_{\text{env}}(s_{\text{seed}}, F, E), \tag{4}$$

where $\mathcal{G}_{\text{env}}$ denotes that the LLM-based scenario generator adapts to specific environments. Environment-specific prompts assist the LLM in understanding its tasks and target environment, guiding the generation of new scenarios. During testing, the seed scenario s_{seed}, expert experience E, and historical testing feedback F are automatically incorporated into the prompts. s_{seed} serves as a reference for scenario generation, while E enhances the LLM's reasoning capabilities with human expertise. Historical feedback F is crucial, as it enables the LLM to generate scenarios through online optimization, allowing it to identify more critical scenarios relevant to the policies under test.

1) Prompt Engineering: A scenario generation pipeline powered by structured prompts is proposed in this paper to drive the LLM-based scenario generator. As illustrated in Fig. 3, the LLM generates critical testing scenarios through a structured process that includes task assignment, knowledge embedding, and a series of explicit reasoning steps.

At the outset of the prompt, to facilitate the LLM's rapid adaptation from a general-purpose language model to a domain-specific scenario generator, we assign it the role of a "test scenario generator" and specify that its task is to mutate the state of a given scenario in a way that induces failure in the target policy.

Subsequently, the LLM is equipped with both static and dynamic knowledge through structured embedding. The static **scenario information** encompasses environment-specific background information that remains constant throughout the testing process, such as the scenario description, the meaning of the initial state, and the constraints governing scenario generation. The dynamic **input message** (i.e., s_{seed}, E, and F) is updated during testing to reflect ongoing results and accumulated experience. The initial state of the seed scenario is transformed into the format described in the scenario information.

Bad cases encountered during testing are considered as F and incorporated into the prompt. F is represented as structured feedback consisting of the initial states of previously generated bad cases, rather than textual summaries or full

TABLE I: Details of the key elements for prompt engineering

Type	Key Element	Description
Instruction	Role Assignment	Assign the LLM a role (e.g., Test scenario generator)
	Task Introduction	Define the task of LLM
Scenario Information	Overview	Offer a concise and accurate description of the target environment (e.g., the agent(s)' tasks, the definitions of crash or failure, and other essential details)
	Entity Information	Cover the common characteristics (e.g., number, type, physical properties, and interactions) of the entities within the environment (e.g., agent(s) controlled by the target policies, and any objects that may interact with the agent(s), such as obstacles)
	State Description	Explain the meaning of the scenario's parameters provided in the subsequent input message
	Constraints	List the principles that must be followed during scenario generation (e.g., the constraints that ensure the newly generated scenarios are valid and solvable)
Input Message	Seed Scenario	Offer a set of parameters representing the seed scenario (e.g., the initial states)
	Testing Feedback	Offer any relevant information acquired during the testing process
	Expert Experience	Offer the knowledge and insights from human testers
Scenario Generation	Scenario Analysis	Guide the LLM to analyze the given seed scenario and highlight the key information (e.g., the agents' states and the relationships between the entities)
	Evolution Prediction	Guide the LLM to predict the future evolution of the scenario (e.g., the agents' trajectories and interactions between the entities)
	Challenge Analysis	Guide the LLM to identify ways to cause the decision-making task to fail and provide the corresponding idea
	Plan Generation	Guide the LLM to generate a plan for modifying the seed scenario
	Plan Execution	Guide the LLM to execute the plan accurately and generate a new scenario in the same format as the input

trajectories. In this paper, three criteria are used to classify a new scenario as a bad case: *i) Insufficient Challenge*: This type of bad case is recorded when there is a significant increase in reward after generation, i.e., $r_{\text{new}} - r_{\text{seed}} > \mathcal{T}_r$, where $\mathcal{T}_r$ is a fixed threshold. This indicates that the LLM is not mutating the given scenario in a way that increases its difficulty. *ii) Invalidity*: This type of bad case is recorded when the new scenario breaks the constraints or is unsolvable even by an optimal policy. *iii) Excessive Modification*: This type of bad case is recorded when the difference between the new scenario and the seed scenario exceeds a specified threshold $\mathcal{T}_s$, i.e., $||s_{\text{new}} - s_{\text{seed}}|| > \mathcal{T}_s$. This criterion is applied when the tester expects the LLM to generate scenarios that are similar to the seed scenario. Whenever the LLM generates a scenario that violates any predefined criterion, it is recorded as a bad case. In subsequent iterations, if the same seed scenario reappears, the associated bad cases are injected into the prompt as feedback. To reduce overhead, only the initial state of each bad case, together with its associated violated criterion, is included. When the number of bad cases becomes large, a fixed-size subset is sampled for each prompt, ensuring that the feedback size remains bounded and preventing prompt overflow or degradation due to long context.

The expert experience E in this pipeline follows the format of mutation plans. With this format, the LLM can easily reference expert experience and generate its own plan based on the expert's plan. In practice, the design of expert experience depends on task characteristics and the tester's domain knowledge, and its effectiveness is typically evaluated empirically by comparing the generated scenarios with and without the provided guidance. In this context, the mutation plan can be very simple, requiring minimal effort from the tester (e.g., "slightly mutate the seed to make the scenario more challenging"). Alternatively, it can be more intricate, based on prior knowledge and experience.

To guide the LLM's reasoning during scenario generation, we employ COT [20] prompting. A well-designed COT of scenario reasoning is used to enhance the model's ability to interpret and manipulate the scenario by guiding it through a structured reasoning process. This includes analyzing the current scenario, predicting its evolution, identifying potential challenges based on those predictions, and formulating a mutation plan accordingly—integrating both static and dynamic knowledge.

Finally, the LLM is instructed to break the scenario generation task down into a series of achievable sub-tasks, facilitating the execution of the proposed mutation plan. The output is formatted consistently with the input state, ensuring easy extraction during post-processing. A complete prompt example, including the representations of s_{seed}, E, and F, is provided in Appendix A.

2) Prompt Template: To minimize the effort required for prompt engineering, we analyze the characteristics of decision-making tasks and summarize the proposed pipeline of prompt engineering. We refine and identify several key elements to include in the prompt, categorizing them into four types of tokens: instruction, scenario information, input message, and scenario generation. Table I provides detailed descriptions of these key elements.

Each type of token provides the LLM with the necessary information and guidance, enabling it to generate new scenarios from seed scenarios. Specifically, at the outset of the prompt, instruction tokens assign a role to the LLM and clearly define the task, helping the LLM understand both its role and the task at hand. Next, detailed scenario information is provided to the LLM to help it better understand the key aspects of the target environment. These tokens convey general details about the environment, which remain constant throughout the

```
# Test Scenario Generator
Role: You are a professional test scenario generator, and you need to create
scenarios that are challenging to the test object to make its task fail.
In <environment description >, you need to design a mutation plan to slightly
change <the variable parameters> in the initial state so that <crash definition,
e.g. the vehicles collide with each other>

## Scenario Information
- Overview: <concise and appropriate description>
- Entity Info: <common information of entities, e.g. number, size, ...>
- State description: <the meaning of variable parameters>
- Constraints: < several principles that must be followed>
- <other information>

## Input
### Current Scenario:
<state of the current scenario>
### Mutation Plans for Selection
(Here are some mutation plans from which you should choose one or some to
help generate a new initial state.)
i. <plan i>
...
### Bad Cases
- Bad Case i: (Reason: <the reason why this case is bad, e.g. can not lead to a
failure>)
<details of bad case i>

## Output
(You need to follow the following steps to output, IMPORTANT!)
### 1. Chain of Thoughts
(Let's think step by step following the Chain of Thoughts, you should answer all
points mentioned below)
- <question about analyzing the given scenario>
- <question about predicting the future evolution of the given scenario>
- <question about thinking how to make the target algorithm fail>
- <question about generating a new mutation plan>
- <other questions>

### 2. Mutate the initial state
- If you are planning to mutate the initial state (The new state should match the
Constraints, and do not mutate too much), remember that what you should
mutate are <the variable parameters>. You are going to make <crash definition>
and make the scenario more challenging.(IMPORTANT)
- Please ensure that the new state match the description in the Chain of Thoughts
section and the Constraints.(IMPORTANT)
- <other necessary demands>

In the Chain of Thoughts you have design a mutation plan, you need to
implement it accurately. Let's think step by step, break this task down into a
number of achievable sub-tasks, and implement them step by step.
2.1. Output the step by step calculate as the following format:
```
step i : <necessary calculation of this step>
```
2.2. Finally, based on 2.1, the new state must be generated as the following
python format without any other information or annotation. (MOST
IMPORTANT)
<format of output>
### 3. Explain the new state
```

Fig. 4: Demonstration of a prompt template. *Instruction* (Role) provides an overview of the LLM's task and the target environment. *Scenario Information* (##Scenario Information) outlines the common and fixed information of the target environment. *Input Message* (##Input) includes the variable information during testing, such as the seed scenario, feedback, and expert experience. *Scenario Generation* (##Output) guides the LLM to generate and output a new scenario in a specified format.

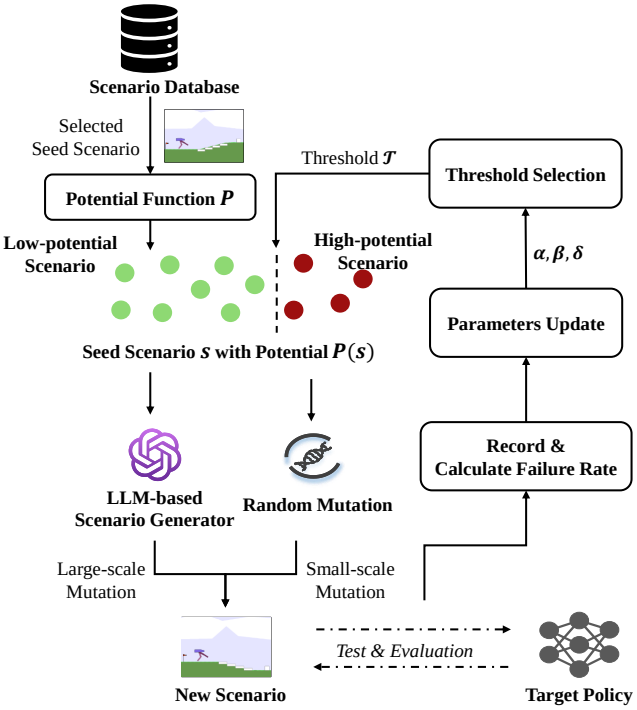


Fig. 5: Illustration of the multi-scale generation strategy.

testing process. Furthermore, the input message tokens contain available data that vary as the scenario changes, corresponding to s_{seed}, F, and E in Eq. (4). These tokens serve as input interfaces for available data. Based on the background information and available data, the LLM is tasked with thoroughly understanding the scenario and generating one that meets the testing requirements. To facilitate this, the scenario generation tokens guide the LLM through a gradual reasoning process, ultimately producing an output in the expected format.

By integrating the key elements outlined in Table I, a prompt can be efficiently designed without introducing excessive workload. Various prompting techniques and styles can be combined to create the prompt, and Fig. 4 presents the prompt template used in this paper.

C. Multi-scale Generation Strategy

Despite great model capacities, LLMs are known to struggle with fine-grained and complex tasks, such as those that require high computational precision. In the context of critical scenario generation, this limitation results in poor performance when fine adjustments are made to seed scenarios, undermining their overall effectiveness. In response to these issues, we propose a novel multi-scale scenario generation strategy which enhances efficiency while conserving valuable testing resources. The workflow is illustrated in Fig. 5.

1) Motivation: LLM-driven scenario generation establishes a direct link between the generator and the target environment, making it highly targeted and efficient. However, prior studies have shown that LLMs remain limited in numerical reasoning and in generating precise parameter values within specified ranges [19]. LLMs inherently exhibit a discretization charac-

teristic in mutation amplitude, which limits their capacity for local exploration. This limitation arises because LLMs tend to output numbers that are contextually common or typical. For instance, when asked to modify the number 1.5 within a range of 0 to 10, an LLM is more likely to produce values such as 1 or 2, rather than more granular adjustments like 1.71. As a result, LLMs show limited sensitivity to small distinctions in numerical values and struggle to capture subtle differences between scenarios. Consequently, such generators are not well-suited for effectively utilizing near-failure scenarios, which are only one step away from critical situations. This limitation is particularly problematic near the failure boundary, where even small parameter perturbations may lead to qualitatively different outcomes, as observed in prior work such as MDP-Fuzz [41]. In contrast, a small and random mutation can provide fine-grained variation to the seed scenario, resulting in better local exploration ability. Accordingly, we propose a multi-scale generation strategy that enables large-scale mutations using the LLM in general scenarios, while applying small-scale random mutations in near-failure scenarios.

2) Scenario Potential Analysis: In this work, we categorize the mutations applied to the seed scenarios based on the estimated potentials. Specifically, each scenario is characterized by its distance from critical scenarios, which reflects its potential to escalate into a critical scenario following random perturbations. A higher potential indicates a greater likelihood of transitioning into a critical scenario with minor perturbations. Firstly, we denote *High-potential Scenarios* as those that are near-failure and more likely to evolve into critical scenarios, potentially resulting in a crash. Then, for quantitative evaluations, we introduce a potential prediction function P, where the potential of a given scenario is calculated as $P(s)$. In this way, such binary classification can be flexibly proceeded through a threshold $\mathcal{T}$, such that the set of high-potential scenarios is represented as

$$\mathcal{H} = \{s | P(s) \geq \mathcal{T}\}, \tag{5}$$

while the set of *Low-potential Scenarios* is as

$$\mathcal{L} = \overline{\mathcal{H}} = \{s | P(s) < \mathcal{T}\}. \tag{6}$$

Note that the cumulative reward r from the *Scenario Evaluation* module in Section IV-A is highly relevant to the task completion quality of the tested policy within each scenario. Thus, we utilize the cumulative reward r to measure the potential function, i.e., $P(s) = -r_s$, where scenarios with lower rewards are regarded as having higher potentials.

3) Generation with Potential Analysis: High-potential scenarios are near-failure, but the LLM-based scenario generator has limitations in fine-tuning these scenarios effectively, prohibiting full exploration of their potentials. In our method, for low-potential scenarios, the LLM-based scenario generator is applied for large-scale and environment-specific searches. Conversely, for high-potential scenarios, we employ random mutation to conduct small-scale and localized explorations, with the procedure described as follows:

$$s_{\text{new}} = \begin{cases} \mathcal{G}_{\text{env}}(s_{\text{seed}}, F, E), & s_{\text{seed}} \in \overline{\mathcal{H}}, \\ s_{\text{seed}} + U(-a, a), & s_{\text{seed}} \in \mathcal{H}, \end{cases} \tag{7}$$

Algorithm 1: Scenario Generation with Multi-scale Generation Strategy

Input: current seed scenario s_{seed} and its potential P_s; P_D which contains potential of each scenario of the scenario database; scenario generator G with parameters α, β, δ; testing feedback F and expert experience E

Output: new scenario after generation s_{new}

if $P_s <$ `Percentile`$(P_D, 1 - G.\alpha)$ **then**
  `// Apply LLM-based generator to low-potential scenarios`
  $s_{\text{new}} \leftarrow G$.LLM_scenario_generation(s_{seed}, F, E);
else
  `// Apply random mutation to high-potential scenarios`
  $s_{\text{new}} \leftarrow G$.random_mutation($s_{\text{seed}}$);
end
if `Failure` (s_{new}) **then**
  `Update_parameters(`G`);`
end
return s_{new}

function `Update_parameters(`G`):`
  $\alpha, \beta, \delta \leftarrow G.\alpha, G.\beta, G.\delta$;
  $rate \leftarrow$ `Calculate_failure_rate()`;
  if $rate < (1 - \delta) * G.last\_rate$ **then**
    `// decay` α `when failure rate decreases`
    $\alpha \leftarrow \alpha * \beta$;
    $G.last\_rate \leftarrow rate$;
  else if $rate > (1 + \delta) * G.last\_rate$ **then**
    `// raise` α `when failure rate increases`
    $\alpha \leftarrow \alpha / \beta$;
    $G.last\_rate \leftarrow rate$;
  end
  $G.\alpha, G.\beta, G.\delta \leftarrow \alpha, \beta, \delta$;
  return

where perturbations are chosen from a uniform distribution with a maximum amplitude of a. In this way, we apply the varying mutation scale to scenarios with different potentials, effectively using the testing information to mitigate the limitations of LLMs.

4) Adaptive Threshold Selection: An appropriate threshold helps identify more critical scenarios while reducing the number of LLM API calls. However, due to the limited understanding of the target environment, directly determining an appropriate threshold for identifying high-potential scenarios is infeasible. Therefore, we normalize using the scenario database, assuming that high-potential scenarios constitute a certain proportion of the database. To accommodate the continuously updated nature of the scenario database, we adaptively select this proportion during the testing process to facilitate determining the threshold.

Algorithm 1 shows the details of multi-scale generation with

adaptive threshold selection. The algorithm initializes with a threshold α, a decay factor $\beta \in (0, 1)$, and an error factor δ. α represents a percentile, where scenarios with potential in the top $\alpha\%$ in the scenario database are considered high-potential scenarios. The core idea of the adaptive algorithm is to calculate the failure rate and update α after discovering new failure cases. Let the new failure rate be denoted by f' and the failure rate calculated the last time α changed be denoted by f. If f' decreases compared to f, this indicates a decline in the quality of the scenario database (i.e., there are fewer high-potential scenarios). In this case, α is decayed by a fixed proportion β. Conversely, if f' increases, α is raised by dividing it by β. To prevent fluctuations in the data, we introduce an error factor δ that tolerates a certain range of variation in the failure rate, such that

$$\alpha' = \begin{cases} \alpha\beta, & f' < (1-\delta)f, \\ \dfrac{\alpha}{\beta}, & f' > (1+\delta)f, \\ \alpha, & \text{otherwise.} \end{cases} \tag{8}$$

In practice, α is expected to be sufficiently low while ensuring it surpasses the optimal threshold, and β can be set within a moderate range to control the magnitude of decay. Through our extensive evaluations, we set $\alpha \approx 20, \beta \approx 0.5$ by default, which can be taken as mild suggestions for practitioners.

With the multi-scale generation strategy, we can leverage the strengths of both LLM-based generation and random mutation. It is worth noticing that this strategy serves as an optional enhancement to the LLM-based scenario generator, with its effectiveness depending on the quality of the scenario database and the definition of $P(\cdot)$. Note that when high-potential scenarios are continuously transformed into critical scenarios and removed without replenishment, α will gradually decrease and approach 0 as the quality of the database deteriorates. At this point, the generator reverts to its original form, ensuring that the efficiency of the strategy is preserved.

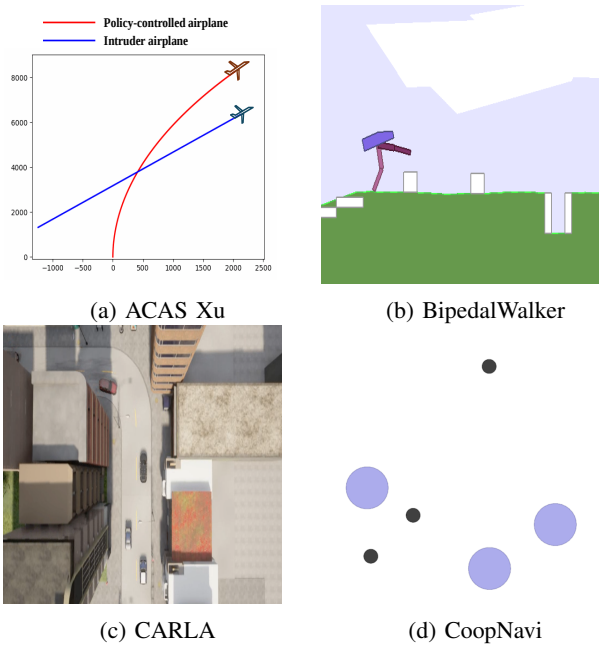
V. Experiments

In this section, a series of experiments are conducted to verify the promising performance of our general LLM-driven adaptive testing framework, LLMTester. The key findings revealed in our evaluations are as follows: *i)* our method identifies more failure cases within the same number of test iterations, thereby confirming the efficient generation of critical scenarios; *ii)* enhanced diversity is achieved in the generated scenarios through the analysis of the identified failure cases; *iii)* ablation studies further validate the effectiveness of the proposed multi-scale generation strategy and prompt pipeline; *iv)* different underlying LLMs demonstrate varying capabilities in generating critical scenarios.

A. Research Questions

The following four research questions are addressed for the main scope of this work. With the extensive experiments, we aim to answer those questions in the subsequent subsections.

- RQ1: Can our proposed LLM-driven adaptive testing method find more critical scenarios that lead to failures in the target policies?
- RQ2: How about the diversity of critical scenarios generated by our LLM-driven approach?
- RQ3: How effective is the proposed multi-scale generation strategy and prompt pipeline?
- RQ4: How do different LLMs affect the scenario generation capabilities of our method?



(a) ACAS Xu (b) BipedalWalker

(c) CARLA (d) CoopNavi

Fig. 6: Target environments of our experiments.

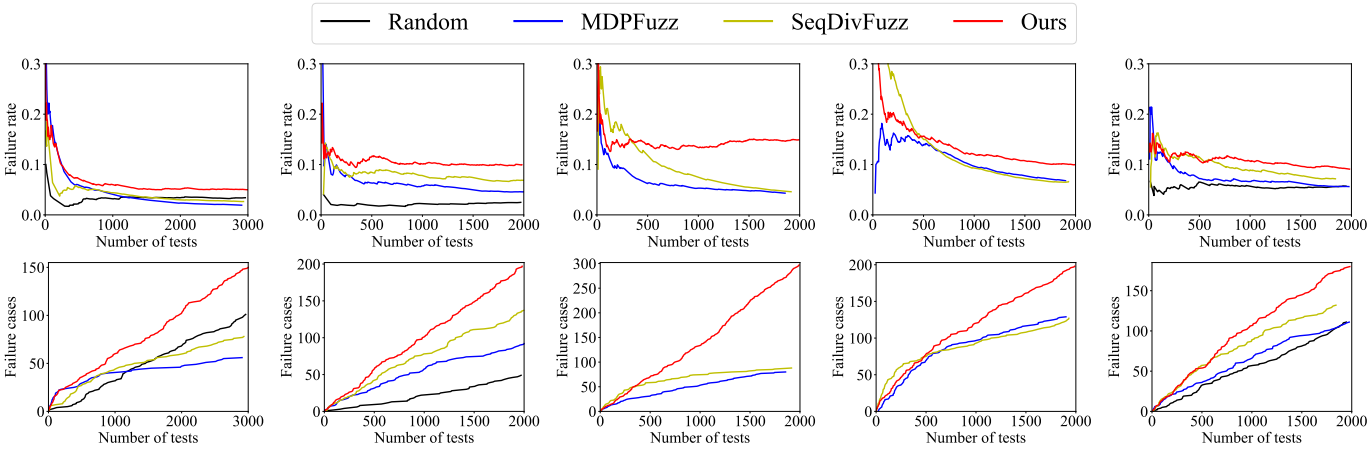
B. Experiment Design

We test 5 distinct policies in 4 different environments, similar to the setup in MDPFuzz [41]. The target environments are illustrated in Fig. 6.

1) Target Policies and Environments: **DNN & ACAS Xu:** ACAS Xu [57] is an airborne collision avoidance system designed for unmanned aircraft. In this study, we test the policy based on a Deep Neural Network (DNN) [58]. The ACAS Xu scenario involves two aircraft: the ownship and the intruder. The goal of the policy is to control the ownship in order to avoid a collision with the intruder, where a collision between the two aircraft is defined as a failure.

RL & BipedalWalker: BipedalWalker is a single-agent environment within OpenAI Gym [59], where the agent is required to navigate bumpy terrain consisting of grass, pits, stumps, and steps. We select TQC [59] from the open-source stablebaseline3 repository [60] as the target policy under test. In this context, a failure is defined as the agent's fall (i.e., its head touches the ground).

RL / IL & CARLA: CARLA [61] is an open-source urban driving simulator designed for autonomous driving. We test well-performing policies based on Reinforcement Learning (RL) [2] and Imitation Learning (IL) [1] in the CARLA simulator. The policies under test are responsible for controlling the steering and throttle of the vehicle to achieve safe autonomous driving. A failure is recorded if the ego vehicle collides with any other entity in the scenario.



(a) ACAS Xu (b) BipedalWalker (c) RL_CARLA (d) IL_CARLA (e) CoopNavi

Fig. 7: Comparison of the number of detected failures and the failure rate between our method and the baselines.

TABLE II: Detailed settings of the experiments.

Model	Environment	#Max frames	#Test	Multi-scale		
				α	β	δ
DNN	ACAS Xu	100	3000	25	0.7	0.1
RL	BipedalWalker	600	2000	10	0.7	0.1
RL	CARLA	100	2000	25	0.7	0.1
IL	CARLA	200	2000	25	0.7	0.1
MARL	CoopNavi	25	2000	20	0.5	0.1

MARL & CoopNavi: CoopNavi [62] is a Multi-Agent Reinforcement Learning (MARL) environment released by OpenAI. In this environment, multiple agents controlled by policies navigate towards landmarks without colliding. We retrain a policy using the released code [62]. A failure in this environment is recorded either when an agent collides with another agent or when the agents fail to reach the designated landmarks within the maximum number of frames.

The target policies in this experiment encompass DNN, RL, IL, and MARL. The state space dimensions across the four environments range from 4 (ACAS Xu) to several hundred (CARLA), thereby covering both low- and high-dimensional cases. This variety is used to assess the scalability of the proposed testing method.

2) Baselines: **Random Testing:** Random testing is a fundamental and straightforward testing approach. As outlined in Section IV-A, the scenario database can be generated through random sampling after parameterizing the variable elements of the environment. Thus, we use randomly sampled scenarios as a baseline for comparison. It is important to note that in the CARLA environment, the positions of vehicles are constrained by the map, and thus are sampled from a pre-constructed scenario library, so that we exclude random testing from the CARLA environment.

MDPFuzz: In the context of Markov Decision Processes (MDPs), an agent makes decisions based on observations from its environment, which is the most typical decision-making problem. MDPFuzz [41] introduces a fuzzy testing framework that follows the process of "seed sampling - seed mutation - corpus update" to efficiently test models solving MDPs. It is the first general testing framework designed for testing models solving MDPs in black-box settings.

SeqDivFuzz: Built upon the MDPFuzz framework, SeqDivFuzz [42] improves testing efficiency by prematurely terminating non-diverse testing cases. It incorporates a diversity inference module based on sequence similarity into the fuzzing process. Leveraging this similarity model, SeqDivFuzz is able to distinguish crash-triggering behaviors among generated testing cases.

3) Experimental Setup: All code for this work is implemented in Python. For MDPFuzz and SeqDivFuzz, we use the original open-source code and modify it to suit our experimental setup. For LLMTester, the implementation details are as follows:

Scenario Database Construction. The scenario database is constructed in two ways, following prior work: (1) random sampling within predefined parameter ranges (used in ACAS Xu, BipedalWalker, and CoopNavi), and (2) direct adoption of pre-defined scenario datasets (used in CARLA, consistent with MDPFuzz).

LLM Configuration. We employ GPT-4o-mini as the backbone LLM due to its balance between generation quality and response efficiency. Four environment-specific prompts are designed based on the unified template described in Section IV-B. A representative request-response example is provided in the appendix.

Evaluation Criteria. Scenario evaluation is conducted along two dimensions: criticality and diversity. Consistent with existing testing literature, we adopt reward as the indicator of criticality and freshness as the indicator of diversity. The reward functions are reused from the original environments and the MDPFuzz implementation to ensure fairness and reproducibility.

The overall experimental settings are summarized in Table II. The *#Max Frames* column specifies the maximum number of time steps allowed per test episode, within which failures are detected. The *#Test* column reports the total number of executed test cases per environment; more than

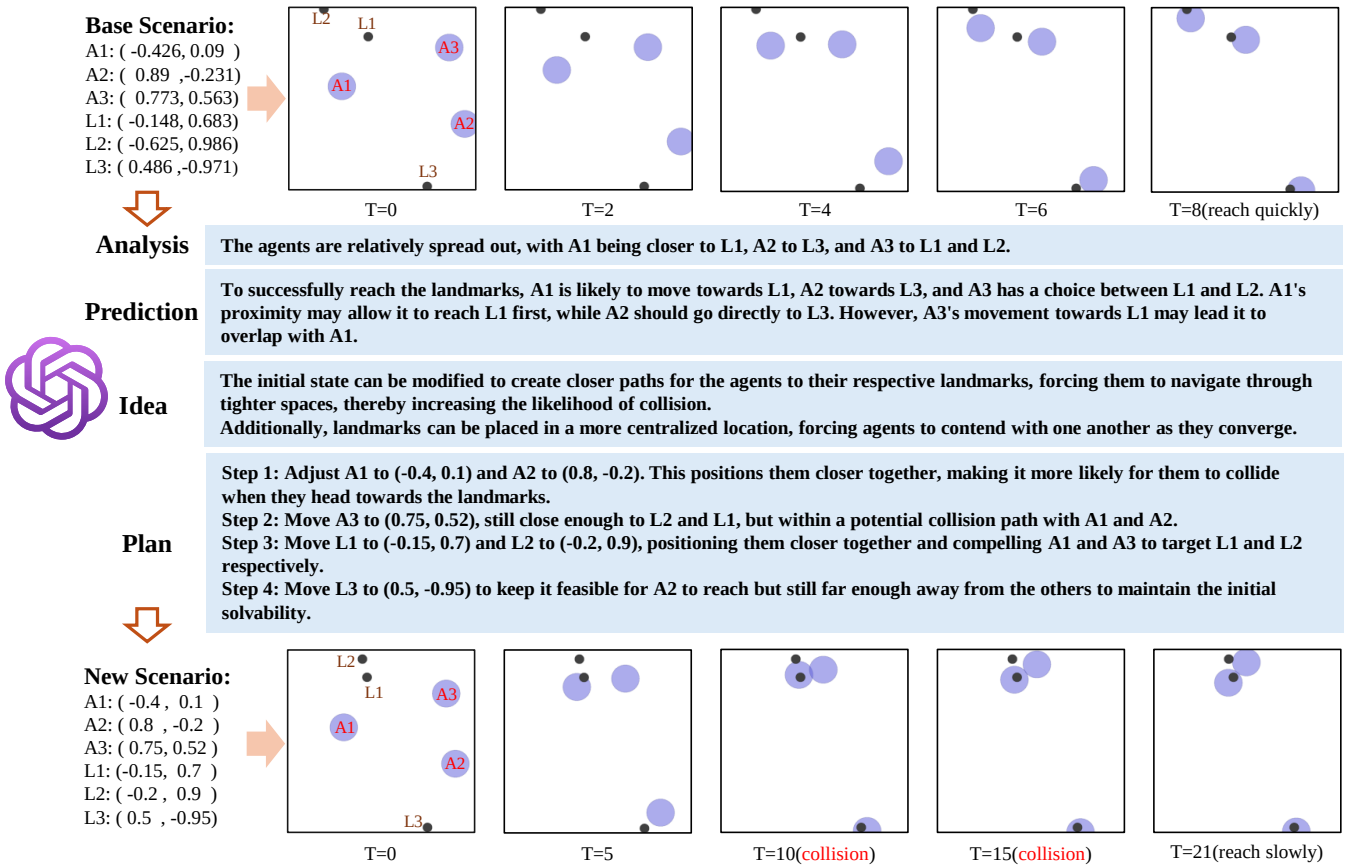


Fig. 8: An illustration of the critical scenarios found by the LLM-based scenario generator.

2000 tests are conducted in each setting to ensure statistical reliability. Notably, for SeqDivFuzz, only fully executed test cases are counted, as its implementation may terminate ongoing tests prematurely. The *Multi-Scale* column lists the hyperparameter configurations used for each target policy.

C. Critical Scenario Generation

In this section, we compare the scenario generation capabilities of the baselines and our method by analyzing the number of failures and the failure rate under the same number of tests. A failure case refers to a scenario in which the decision-making policy fails to complete its task, thus constituting a critical testing scenario. Identified failure cases can be leveraged to analyze the vulnerabilities of the algorithm and to improve its problem-solving capabilities. A higher failure rate and a greater number of failure cases suggest that the method is more efficient in testing and better at generating critical test scenarios. We note that testing efficiency can be evaluated from both fixed testing budget and fixed testing period. In this work, we adopt the fixed-budget setting, as each test may be costly in realistic scenarios, making the number of tests a more practical constraint and a clearer measure of effectiveness in discovering critical scenarios. The experimental results for the baselines and LLMTester, applied to the five tested policies, are presented in Fig. 7. Here, Random testing is not conducted in the CARLA environment, as the scenario database in CARLA is predefined, and random sampling would significantly violate the map constraints governing traffic roads.

As shown by the experiments, our LLM-driven method uncovers more failure-triggering scenarios compared to the best baseline across all five tasks. Specifically, the number of failure cases increases by 48.51%, 43.80%, 237.50%, 53.49%, and 36.36%, respectively. Unlike MDPFuzz and SeqDivFuzz, which generate new test scenarios through random mutation, our method more strategically modifies seed scenarios with a clearer search direction. Furthermore, during the testing process, the failure rates of MDPFuzz and SeqDivFuzz decrease as the scenario database evolves, while our method consistently outperforms random testing and ultimately converges to a higher failure rate, showing the superior stability of our method in generating critical scenarios.

Fig. 8 provides a concrete example of the LLM-driven scenario generation process by illustrating the critical scenarios detected in CoopNavi, along with a segment of the LLM's response generated during the creation of this scenario. Upon receiving the state representing the seed scenario, the LLM first interprets the situation, predicts the subsequent actions of the three agents, and anticipates future interactions among them. Correspondingly, the LLM proposes strategies to increase the scenario's complexity and incorporates them during scenario generation. This process involves developing a detailed and step-by-step plan, ultimately leading to the creation of a more challenging scenario.

TABLE III: The number of failure cases and different states discovered by MDPFuzz, SeqDivFuzz, and our method in different environments.

Environments	Methods	#Failures	#Entire	#Initial	#Terminal
ACAS Xu	MDPFuzz	56	**43**	22	20
	SeqDivFuzz	78	40	**30**	**33**
	Ours	**150**	**43**	22	29
BipedalWalker	MDPFuzz	92	1468	1	49
	SeqDivFuzz	137	1508	1	74
	Ours	**197**	**2033**	1	**98**
RL_CARLA	MDPFuzz	80	1657	34	76
	SeqDivFuzz	88	1619	35	81
	Ours	**297**	**6522**	**164**	**287**
IL_CARLA	MDPFuzz	129	4012	44	105
	SeqDivFuzz	127	3692	49	101
	Ours	**198**	**6398**	**96**	**167**
CoopNavi	MDPFuzz	111	872	109	111
	SeqDivFuzz	132	936	128	132
	Ours	**180**	**1448**	**169**	**175**

In the original seed scenario, the three agents successfully reach their respective landmarks without any significant challenges, completing the task efficiently by the 8^{th} frame. In contrast, in the newly generated scenario, the LLM strategically adjusts the initial positions and repositions the landmarks to introduce obstacles in the agents' movements toward their targets. As a result, agents A1 and A3 collide while navigating to their respective landmarks, causing delays in the completion of the task. Due to multiple collisions, the decision-making policy fails in the new scenario. The LLM-based scenario generator has identified a new critical scenario herein, exposing latent weaknesses in the target policy.

Fig. 8 also shows that the parameters modified by the LLM typically contain only one to two significant figures. This confirms that LLM-generated scenarios often lack the fine-grained precision needed for subtle adjustments, empirically supporting the motivation for our multi-scale generation strategy.

Answer to RQ1: Our LLM-driven adaptive testing framework identifies a greater number of failure cases compared to the baseline methods, demonstrating the superior efficiency of our approach in generating critical scenarios.

D. Diverse Scenario Generation

In this study, we replay all detected failure cases and count the number of distinct states observed by the agent in each target environment. Since the states are continuous, we analyze the maximum and minimum values for each state dimension and divide each dimension into N equal intervals, so that each continuous state is mapped to a discrete cell. Assuming that the observed state has D dimensions, the entire state space is divided into D^N different states. During the replay of the failure cases, the numbers of distinct initial states, terminal states, and states throughout the entire process, denoted as *#Initial*, *#Terminal*, and *#Entire*, respectively, are all reported. *#Initial* and *#Terminal* represent the diversity of initial states leading to failures and the diversity of failure scenes, while *#Entire* reflects the total number of distinct discretized states

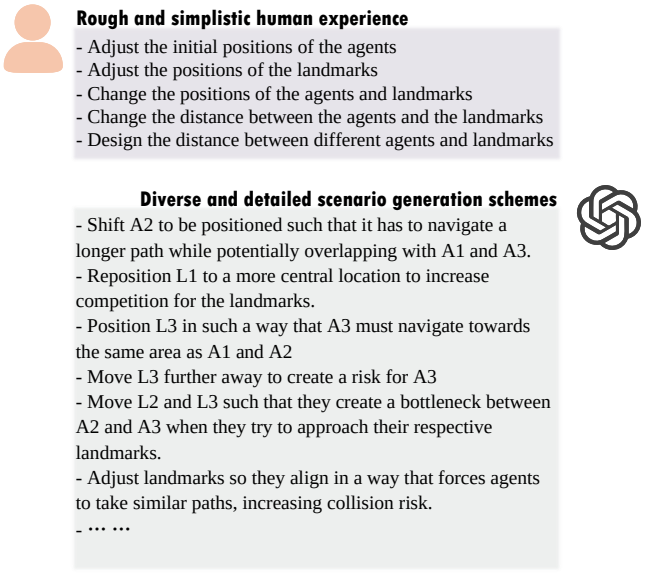


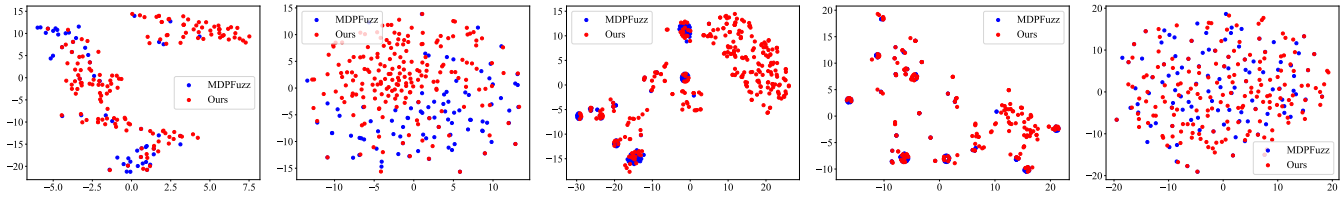
Fig. 9: Human experience and some typical schemes provided by the LLM. The 'A' and 'L' in the figure represent 'Agent' and 'Landmark', respectively.

observed across all timestamps of the scenario, reflecting the overall novelty of the testing scenarios.

The experimental results are presented in Table III. It is noteworthy that only a single initial state is identified in BipedalWalker, due to the fixed initial spawn position and initial terrain (grass). As shown in Table III, our method is able to detect a greater number of distinct states within the same number of tests. In most cases, our method significantly outperforms MDPFuzz and SeqDivFuzz, and even in the worst-case scenario (ACAS Xu), it remains comparable, exhibiting only slightly lower performance than SeqDivFuzz, which benefits from an integrated diversity inference module. A higher number of covered states indicates that the LLM does not focus solely on a few critical testing scenarios, but instead maintains a balance by accounting for the diversity of testing scenarios.

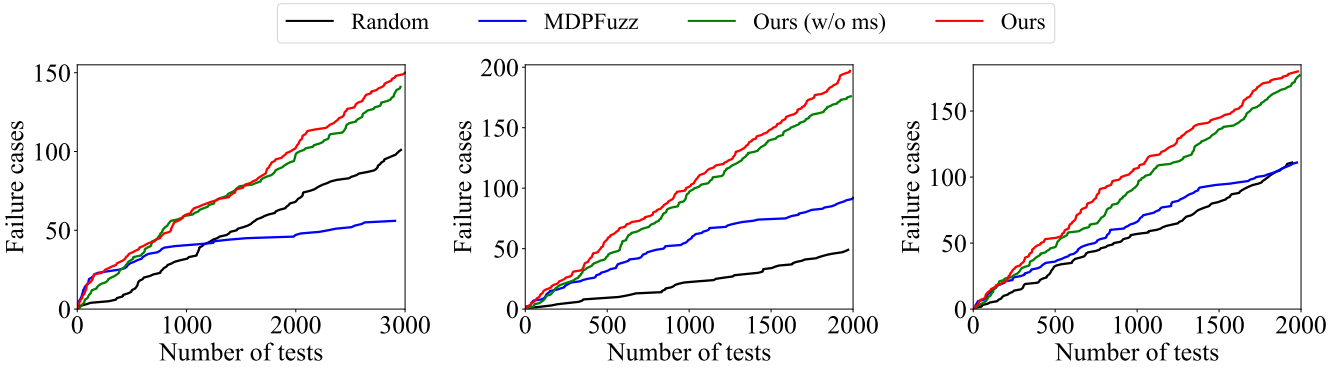
Fig. 9 further investigates both the human-provided experience and a selection of scenario generation schemes proposed by LLMs within the CoopNavi task. The human tester provides only basic and rudimentary experience, lacking substantial prior knowledge. Besides, the LLMs, leveraging their reasoning capabilities and understanding of scenarios, are able to generate more diverse and detailed scenario generation schemes. These varied schemes demonstrate how LLMs approach the creation of challenging test scenarios from multiple perspectives. The results highlight that our method can effectively harness the creative capacity of LLMs to produce a broader range of scenarios. Importantly, this diversity stems from the LLMs' deep comprehension of the scenarios, rather than merely generating data with large variations.

To visually assess the ability of different methods to explore diverse states, we apply t-SNE to project the high-dimensional failure cases into a two-dimensional space. The results, presented in Fig. 10, reveal that our method achieves broader



(a) ACAS_Xu (b) BipedalWalker (c) RL_CARLA (d) IL_CARLA (e) CoopNavi

Fig. 10: t-SNE visualization of discovered failure cases in different environments.



(a) ACAS Xu (b) BipedalWalker (c) CoopNavi

Fig. 11: Detected failures by our method with and without the multi-scale generation strategy.

and more diverse exploration across tasks—particularly in ACAS Xu, BipedalWalker, and CARLA—demonstrating significantly greater coverage of critical regions compared to baseline methods. This enhanced exploration capability can be attributed to the creativity and inherent randomness of the LLM, enabling LLMTester to perform cross-regional searches for critical scenarios. This enhanced exploratory behavior is attributable to the creativity and inherent randomness of the LLM, which enables LLMTester to perform cross-regional searches for critical scenarios. As a result, LLMTester exhibits improved efficiency in uncovering a wide range of failure cases.

Answer to RQ2: Compared to MDPFuzz, our LLM-driven method identifies a comparable or greater number of states in critical scenarios. Unlike other testing methods, which lack human-like intelligence, the LLM generates diverse test scenarios using different generation schemes, drawing on its deep understanding of the scenarios. These results highlight the superior ability of our method to generate diverse scenarios.

E. Ablation Study

In this section, we conduct ablation experiments to further investigate the effectiveness of the proposed multi-scale scenario generation strategy and prompt pipeline. Specifically, we evaluate the contribution of key components in LLMTester by removing them and comparing the resulting number of failure cases.

1) Effectiveness of Multi-scale Generation Strategy: We remove the random mutation module from LLMTester to obtain a variant that relies solely on the LLM for scenario generation, denoted as $Ours\ (w/o\ ms)$.

Fig. 11 presents the testing results across different environments. The scenario generator with the multi-scale strategy consistently detects more failure cases in nearly every iteration compared to the LLM-only variant. This improvement stems from the fact that the multi-scale strategy performs potential analysis on seed scenarios and applies small-scale perturbations to high-potential ones to generate new candidates. Since high-potential scenarios are closely associated with failure cases, and failure-inducing conditions are often unpredictable, incorporating random mutations significantly improves search efficiency. As a result, the number of LLM API calls is reduced by 5.27%, 55.67%, and 16.13%, respectively.

The comparison between $Ours\ (w/o\ ms)$ and Random testing further demonstrates that the LLM-based generator provides a clearer search direction and more effectively identifies critical scenarios. Meanwhile, as promising seed scenarios are progressively transformed into failure-triggering cases, the overall quality of the scenario database deteriorates, leading to a performance degradation in MDPFuzz. In ACAS Xu, MDPFuzz even underperforms Random testing.

Benefiting from the adaptive threshold, our method dynamically regulates the proportion of small-scale mutations and maintains stable performance. To further understand the behavior of the multi-scale strategy, we analyze its sensitivity to key hyperparameters. Specifically, we vary the parameters around the default settings (e.g., $\alpha \approx 20$, $\beta \approx 0.5$) and evaluate the resulting performance, as shown in Fig. 12.

The results on ACAS Xu and CoopNavi indicate that performance remains stable under moderate parameter variations.

TABLE IV: The rate of LLM-based generation and final α in BipedalWalker with different hyperparameter configurations.

α - β - δ	5-0.7-0.1	10-0.7-0.1	10-0.5-0.1	20-0.5-0.1
Ratio (%)	88.86	44.33	59.92	24.59
Final α	0.84	10	5	20
Failure cases	218	197	192	141

Small perturbations in α and β do not lead to significant differences, suggesting that the proposed strategy is reasonably robust to hyperparameter selection. However, in BipedalWalker, we observe a more pronounced sensitivity to α. Specifically, reducing α from 10 to 5 leads to improved performance, as it enforces stricter selection of high-potential scenarios and yields a more balanced sampling process. As shown in Table IV, with $\alpha = 5$, the proportion of high-potential scenarios (88.86%) is closer to that in other environments (94.73% in ACAS and 83.87% in CoopNavi), highlighting the importance of maintaining an appropriate proportion for effective LLM-guided generation.

Overall, these findings suggest that while the multi-scale generation strategy is not overly sensitive to hyperparameters in most environments, its performance can still benefit from moderate tuning in specific cases. Therefore, the default settings should be regarded as general guidelines rather than universally optimal choices.

2) Effectiveness of Prompt Pipeline: Fig. 13 illustrates the number of failure cases observed across three environments following the selective removal of individual components from the prompt: expert knowledge, testing feedback, and Chain-of-Thought (CoT) reasoning – denoted as "w/o Experience," "w/o Feedback," and "w/o CoT," respectively.

As shown in Fig. 13, compared to the complete prompt, the removal of any single component results in a noticeable reduction in failure cases, highlighting the individual contribution of each element to the prompt's overall effectiveness. Among these components, COT has the most significant impact, resulting in an average 22.03% decrease in failure cases, followed by testing feedback, which yields an average reduction of 5.88%. The influence of expert knowledge, however, varies across environments. Specifically, removing expert knowledge ("w/o Experience") results in a slight decrease in CoopNavi, and a substantial decrease in BipedalWalker, but a slight increase in failures in ACAS Xu. This variability is likely due to differences in the quality and relevance of the expert knowledge provided. In ACAS Xu, for example, the expert knowledge is relatively coarse – e.g., "Change the speed of the ego so that the two planes are liable to collide" – which may introduce ambiguity and confuse the LLM. In contrast, BipedalWalker benefits from more detailed and context-specific knowledge, such as: "Multiple non-grass fields in a row are stumps or pits that walkers need to constantly step over," which provides clearer cues for scenario understanding and generation. Overall, these results indicate that the impact of expertise knowledge scales with its level of detail and informativeness, where more structured and detailed knowledge yields stronger improvements, while coarse guidance provides limited benefits.

The experimental results demonstrate the importance of each component of the prompts. Effective expert experience, testing feedback, and COT are helpful for LLMs to generate critical test scenarios better.

Answer to RQ3: The ablation experiments validate the effectiveness of our design. First, the multi-scale generation strategy significantly improves the efficiency of the LLM-driven adaptive testing process while simultaneously reducing the consumption of testing resources. Second, each component of the structured prompt enhances LLM-based scenario generation by providing complementary information from distinct perspectives.

TABLE V: Comparison of the number of failures and the number of different states across different underlying LLMs.

Environments	LLM	#Failures	#Entire	#Initial	#Terminal
ACAS Xu	GPT-4o-mini	150	43	22	29
	GLM-4-Air	86	38	19	22
	DeepSeek-V3	113	**62**	**29**	**33**
	Gemini-2.5-Flash	126	49	23	28
	Claude-Sonnet-4.5	**191**	55	26	**33**
BipedalWalker	GPT-4o-mini	197	**2033**	1	98
	GLM-4-Air	135	1847	1	67
	DeepSeek-V3	**216**	1977	1	**106**
	Gemini-2.5-Flash	124	1608	1	68
	Claude-Sonnet-4.5	158	1735	1	87
CoopNavi	GPT-4o-mini	180	1448	169	175
	GLM-4-Air	172	1299	159	170
	DeepSeek-V3	315	2136	**262**	290
	Gemini-2.5-Flash	473	2151	221	347
	Claude-Sonnet-4.5	**680**	**2729**	254	**438**

F. Comparison of Different LLMs

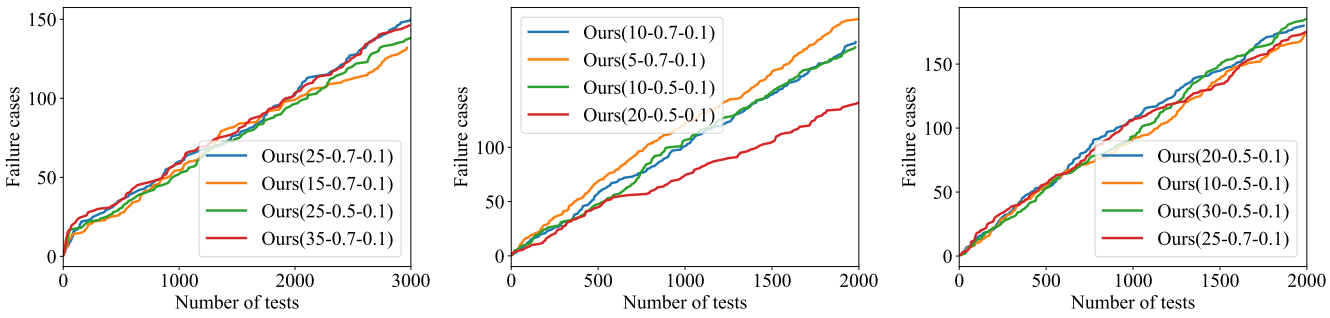
We further analyze the impact of different LLMs on the performance of our LLM-driven method. In the previous experiments, we used GPT-4o-mini, which balances efficiency and response speed. In this study, we additionally evaluate GLM-4-Air, DeepSeek-V3, Gemini-2.5-Flash, and Claude-Sonnet-4.5, which show varied performance across different benchmarks.

As shown in Table V, different LLMs exhibit distinct performance across environments, and all evaluated models generally achieve good performance in supporting the testing process.

The models demonstrate their respective strengths. In the ACAS Xu environment, Claude-Sonnet-4.5 identifies the most failure cases, demonstrating its capability in generating challenging scenarios. Notably, Claude-Sonnet-4.5 and GPT-4o-mini, although discovering more failure cases, are slightly less effective than DeepSeek-V3 in covering diverse discrete states. This suggests that in ACAS Xu, they tend to follow reasoning patterns focused on generating critical scenarios, which are effective but somewhat limited in diversity and creativity.

In CoopNavi, Claude-Sonnet-4.5 now achieves the best performance, while Gemini-2.5-Flash also shows strong results, indicating that these models are better suited to the demands of their respective environments. In BipedalWalker, all models achieve satisfactory performance, with DeepSeek-V3 slightly ahead.

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2026.3703032



(a) ACAS Xu (b) BipedalWalker (c) CoopNavi

Fig. 12: The number of failure cases discovered by LLMTester with different configurations of hyperparameters

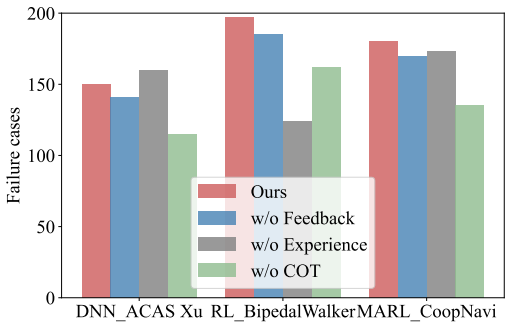


Fig. 13: Comparison of the number of failures under different prompts.

These comparative experiments show that the choice of underlying LLM significantly influences testing effectiveness. Differences in reasoning capabilities across LLMs, combined with the stochastic nature of target policies and environments, lead to environment-dependent variations in performance. Consequently, the effectiveness of a given LLM in scenario generation can vary notably across applications. Importantly, our results confirm that the framework can leverage the strengths of different LLMs while maintaining adaptability across environments.

Answer to RQ4: Different underlying LLMs demonstrate varying capabilities in generating critical scenarios. By preserving the distinct characteristics of each model, our framework exhibits strong adaptability and holds considerable potential for further enhancement as LLMs continue to advance in intelligence.

VI. Discussions & Threats To Validity

In this section, we discuss possible threats to the validity of our research, together with the solutions for mitigation, and possible directions for future work.

Workload of Prompt Design. In prompt engineering, the primary manual effort lies in describing the target environment and crafting the COT, both of which require a solid understanding of the target policy and the environment. For mitigation, we streamline the process by using a prompt template specifically tailored to decision-making problems. This approach allows for the efficient creation of effective prompts with only a basic understanding of the policy and environment. Nevertheless, future work could explore incorporating more automated prompt generation techniques to further improve the efficiency and scalability of our framework.

Performance of the Underlying LLM. As discussed in Section V, the performance of different LLMs can significantly influence their ability to generate testing scenarios. The efficiency of critical scenario generation depends on the LLM's capacity to comprehend the scenario, integrate expert knowledge, and incorporate feedback. Similarly, the diversity of generated scenarios reflects the LLM's creativity, which places specific demands on the intelligence level of the chosen model. In addition, the typical response time for LLM API calls, often measured in seconds, can limit testing speed, particularly for simpler tasks in simulation environments. In this paper, we propose a well-designed framework that leverages various forms of guidance, such as sensitivity, freshness, and cumulative reward, to address these challenges. To mitigate the impact of LLM response times, we introduce a multi-scale generation strategy that reduces the number of LLM API calls by incorporating random mutations. These challenges underscore the significant growth potential of the proposed LLM-driven framework, especially as LLM technologies continue to advance. In addition to the anticipated advances in LLMs, exploring technologies that can more effectively stimulate LLMs' capabilities remains a valuable avenue for investigation.

Dimension of the State Space. In many complex environments, the presence of numerous scenario parameters results in a high-dimensional state space. This high dimensionality primarily impacts the LLM's ability to comprehend the given scenario, posing a challenge to its contextual understanding and potentially affecting testing performance. It is worth noting that current mainstream LLMs already demonstrate strong reasoning capabilities and show promising potential in identifying key scenario parameters. Our experiments, which span both low- and high-dimensional scenarios, provide empirical support for this observation. Nonetheless, when the application environment becomes excessively complex or the dimensionality of the state space increases beyond the model's comprehension capacity, threats to validity may arise. To address this issue, future research could explore the use of

more advanced LLMs or develop preprocessing techniques for extracting key scenario parameters to facilitate understanding.

VII. Conclusion

In this work, inspired by the problem-solving capabilities of LLMs, we introduce LLMs as an innovative solution for testing general decision-making policies, presenting a universal and efficient adaptive testing framework, LLMTester. Unlike other methods, by harnessing the intelligence of LLMs, LLMTester is capable of accounting for the specific characteristics of the environment under test, while also flexibly integrating human expertise and testing feedback. An efficient pipeline, incorporating a carefully designed prompt template, is developed to generate critical testing scenarios through environment-specific searches. Recognizing the inherent limitations of LLMs in making fine-grained adjustments, we propose a multi-scale generation strategy that further improves testing efficiency to overcome this challenge. Our extensive experimental results demonstrate that the proposed method outperforms baseline approaches by identifying a greater number of failure cases while maintaining diversity in the generated scenarios. Furthermore, we show that our method effectively mitigates the limitations of LLMs, exhibiting significant robustness and considerable promise for future advancements.

Future work will involve refining the existing framework to facilitate a more automated testing process and better leverage the capabilities of the underlying LLM. Furthermore, exploring more effective evaluation methods for assessing the potential of testing scenarios will also be a focus.

References

[1] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.

[2] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7153–7162.

[3] P. S. Chib and P. Singh, "Recent advancements in end-to-end autonomous driving using deep learning: A survey," *IEEE Transactions on Intelligent Vehicles*, 2023.

[4] H. Pei, J. Zhang, Y. Zhang, H. Xu, and L. Li, "Self-organized routing for autonomous vehicles via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 1, pp. 426–437, 2024.

[5] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[6] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, "A survey of embodied ai: From simulators to research tasks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 2, pp. 230–244, 2022.

[7] Y. Shi, H. Pei, L. Feng, Y. Zhang, and D. Yao, "Towards fault tolerance in multi-agent reinforcement learning," 2024. [Online]. Available: https://arxiv.org/abs/2412.00534

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[9] Z. He, J. Zhang, D. Yao, Y. Zhang, and H. Pei, "Adversarial generation of safety-critical lane-change scenarios for autonomous vehicles," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, 2023, pp. 6096–6101.

[10] S. Feng, H. Sun, X. Yan, H. Zhu, Z. Zou, S. Shen, and H. X. Liu, "Dense reinforcement learning for safety validation of autonomous vehicles," *Nature*, vol. 615, no. 7953, pp. 620–627, 2023.

[11] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "Survey on scenario-based safety assessment of automated vehicles," *IEEE access*, vol. 8, pp. 87 456–87 477, 2020.

[12] M. Biagiola and P. Tonella, "Boundary state generation for testing and improvement of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2024.

[13] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 1016–1026.

[14] K. D. Julian, R. Lee, and M. J. Kochenderfer, "Validation of image-based neural network controllers through adaptive stress testing," in *2020 IEEE 23rd international conference on intelligent transportation systems (ITSC)*. IEEE, 2020, pp. 1–7.

[15] H. Sha, Y. Mu, Y. Jiang, L. Chen, C. Xu, P. Luo, S. E. Li, M. Tomizuka, W. Zhan, and M. Ding, "Languagempc: Large language models as decision makers for autonomous driving," *arXiv preprint arXiv:2310.03026*, 2023.

[16] H. Sun, Y. Zhuang, L. Kong, B. Dai, and C. Zhang, "Adaplanner: Adaptive planning from feedback with language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[17] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[18] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," *arXiv preprint arXiv:2402.07927*, 2024.

[19] C. Lu, P. Valle, J. Wu, E. Isaku, H. Sartaj, A. Arrieta, and S. Ali, "Foundation models for software engineering of cyber-physical systems: the road ahead," *arXiv preprint arXiv:2504.04630*, 2025.

[20] 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.

[21] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[22] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[23] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*. IEEE, 2018, pp. 100–111.

[24] A. Zolfagharian, M. Abdellatif, L. C. Briand, M. Bagherzadeh, and S. Ramesh, "A search-based testing approach for deep reinforcement learning agents," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, pp. 3715–3735, 2023.

[25] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[26] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 2018, pp. 132–142.

[27] J. Duan, F. Gao, and Y. He, "Test scenario generation and optimization technology for intelligent driving systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 1, pp. 115–127, 2020.

[28] S. Feng, X. Yan, H. Sun, Y. Feng, and H. X. Liu, "Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment," *Nature communications*, vol. 12, no. 1, p. 748, 2021.

[29] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1–7.

[30] P. Du and K. Driggs-Campbell, "Finding diverse failure scenarios in autonomous systems using adaptive stress testing," *SAE International Journal of Connected and Automated Vehicles*, vol. 2, no. 12-02-04-0018, pp. 241–251, 2019.

[31] H. Delecki, M. Itkina, B. Lange, R. Senanayake, and M. J. Kochenderfer, "How do we fail? stress testing perception in autonomous vehicles," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 5139–5146.

[32] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali, "Learning configurations of operating environment of autonomous vehicles to

maximize their collisions," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 384–402, 2022.

[33] X. Ma, Y. Wang, J. Wang, X. Xie, B. Wu, Y. Yan, S. Li, F. Xu, and Q. Wang, "Diversity-oriented testing for competitive game agent via constraint-guided adversarial agent training," *IEEE Transactions on Software Engineering*, 2024.

[34] D. Zhao, H. Lam, H. Peng, S. Bao, D. J. LeBlanc, K. Nobukawa, and C. S. Pan, "Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques," *IEEE transactions on intelligent transportation systems*, vol. 18, no. 3, pp. 595–607, 2016.

[35] S. Feng, Y. Feng, C. Yu, Y. Zhang, and H. X. Liu, "Testing scenario library generation for connected and automated vehicles, part i: Methodology," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1573–1582, 2020.

[36] J. Yang, H. Sun, H. He, Y. Zhang, H. X. Liu, and S. Feng, "Adaptive safety evaluation for connected and automated vehicles with sparse control variates," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 2, pp. 1761–1773, 2023.

[37] X. Yan, Z. Zou, S. Feng, H. Zhu, H. Sun, and H. X. Liu, "Learning naturalistic driving environment with statistical realism," *Nature communications*, vol. 14, no. 1, p. 2037, 2023.

[38] Z. Zhong, G. Kaiser, and B. Ray, "Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1860–1875, 2022.

[39] M. Cheng, Y. Zhou, and X. Xie, "Behavexplor: Behavior diversity guided testing for autonomous driving systems," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 488–500.

[40] Y. Huai, S. Almanee, Y. Chen, X. Wu, Q. A. Chen, and J. Garcia, "scenorita: Generating diverse, fully-mutable, test scenarios for autonomous vehicle planning," *IEEE Transactions on Software Engineering*, 2023.

[41] Q. Pang, Y. Yuan, and S. Wang, "Mdpfuzz: testing models solving markov decision processes," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 378–390.

[42] K. Wang, Y. Wang, J. Wang, and Q. Wang, "Fuzzing with sequence diversity inference for sequential decision-making model testing," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 706–717.

[43] Z. Li, X. Wu, D. Zhu, M. Cheng, S. Chen, F. Zhang, X. Xie, L. Ma, and J. Zhao, "Generative model-based testing on decision-making policies," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 243–254.

[44] X. Ma, Y. Wang, J. Wang, X. Xie, B. Wu, S. Li, F. Xu, and Q. Wang, "Enhancing multi-agent system testing with diversity-guided exploration and adaptive critical state exploitation," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1491–1503.

[45] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, 2024.

[46] S. Yu, C. Fang, Y. Ling, C. Wu, and Z. Chen, "Llm for test script generation and migration: Challenges, capabilities, and opportunities," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. IEEE, 2023, pp. 206–217.

[47] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring llm-based general bug reproduction," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2312–2323.

[48] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models," in *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*, 2023, pp. 423–435.

[49] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, Z. Tian, Y. Huang, J. Hu, and Q. Wang, "Testing the limits: Unusual text inputs generation for mobile app crash detection with large language model," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[50] J. Zhang, C. Xu, and B. Li, "Chatscene: Knowledge-enabled safety-critical scenario generation for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 459–15 469.

[51] Y. Wei, Z. Wang, Y. Lu, C. Xu, C. Liu, H. Zhao, S. Chen, and Y. Wang, "Editable scene simulation for autonomous driving via collaborative llm-agents," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 15 077–15 087.

[52] X. Li, E. Liu, T. Shen, J. Huang, and F.-Y. Wang, "Chatgpt-based scenario engineer: A new framework on scenario generation for trajectory prediction," *IEEE Transactions on Intelligent Vehicles*, 2024.

[53] C. Chang, S. Wang, J. Zhang, J. Ge, and L. Li, "Llmscenario: Large language model driven scenario generation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2024.

[54] Q. Lu, X. Wang, Y. Jiang, G. Zhao, M. Ma, and S. Feng, "Multimodal large language model driven scenario testing for autonomous vehicles," *arXiv preprint arXiv:2409.06450*, 2024.

[55] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2018, pp. 2575–2582.

[56] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[57] M. Marston and G. Baca, "Acas-xu initial self-separation flight tests," NASA, Tech. Rep., 2015.

[58] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 598–608, 2019.

[59] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5556–5566.

[60] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rlbaselines3-zoo, 2020.

[61] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[62] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

Weichao Xu received the B.S. degree from Department of Automation, Tsinghua University, Beijing, China, in 2023. He is currently working towards the M.S. degree with the Department of Automation, Tsinghua University, Beijing, China. His current research interests include intelligent vehicle-infrastructure cooperative systems, autonomous driving perception and intelligent system testing.

Huaxin Pei received the B.S. degree from University of Electronic Science and Technology of China, China, in 2017, and earned the Ph.D. degree in the control science and engineering with the Department of Automation, Tsinghua University, China, in 2023. He is currently an assistant researcher at QiYuan Lab. His current research interests include intelligent system testing, generative AI, and autonomous driving.

Jingxuan Yang received the Bachelor's degree from the School of Mechanical Engineering and Automation, Harbin Institute of Technology, Shenzhen, China, in 2020, and the Ph.D. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2025. He is currently a Postdoctoral Research Fellow with the Department of Automation, Tsinghua University, Beijing, China. His current research interests include curse of rarity, autonomous vehicle safety, adaptive testing and adversarial testing.

Yuchen Shi received the B.S. degree from Tsinghua University, Beijing, China, in 2021, where he is currently pursuing the Ph.D. degree with the Department of Automation. His research interests include autonomous driving and multi-agent reinforcement learning.

Yi Zhang (Senior Member, IEEE) received the B.S. in 1986 and M.S. degree in 1988 from Tsinghua University in China, and earned the Ph.D. degree in 1995 from the University of Strathclyde in the UK. He is a professor in the control science and engineering at Tsinghua University with his current research interests focusing on intelligent transportation systems. His active research areas include intelligent vehicle-infrastructure cooperative systems, analysis of urban transportation systems, urban road network management, traffic data fusion and dissemination, and urban traffic control and management. His research fields also cover the advanced control theory and applications, advanced detection and measurement, systems engineering, intelligent system testing, etc.

Qianchuan Zhao (Senior Member, IEEE) received the B.E. degree in automatic control and the B.S. degree in applied mathematics in 1992, and the M.S. and Ph.D. degrees in control theory and its applications from Tsinghua University in 1996. He was a Visiting Scholar with Carnegie Mellon University, Pittsburgh, PA, USA, in 2000; and Harvard University, Cambridge, MA, USA, in 2002. He was a Visiting Professor with Cornell University, Ithaca, NY, USA, in 2006. He is currently a Professor and the Director of the Center for Intelligent and Networked Systems (CFINS), Department of Automation, and BNRist, Tsinghua University. He has published more than 100 research papers in peer-reviewed journals and conferences. His current research interests include the control and optimization of complex networked systems with applications in smart buildings, smart grids, and manufacturing automation.