Generative Model-Based Testing on Decision-Making Policies

Zhuo Li[1], Xiongfei Wu[1], Derui Zhu[2], Mingfei Cheng[3], Siyuan Chen[1], Fuyuan Zhang[1], Xiaofei Xie[3], Lei Ma[56], Jianjun Zhao[1],
[1]*Kyushu University, Fukuoka, Japan* [2]*Technical University of Munich, Munich, Germany*
[3]*Singapore Management University, Singapore* [5]*The University of Tokyo, Tokyo, Japan*
[6]*University of Alberta, Edmonton Canada*

***Abstract*—The reliability of decision-making policies is urgently important today as they have established the fundamentals of many critical applications, such as autonomous driving and robotics. To ensure reliability, there have been a number of research efforts on testing decision-making policies that solve Markov decision processes (MDPs). However, due to the deep neural network (DNN)-based inherit and infinite state space, developing scalable and effective testing frameworks for decision-making policies still remains open and challenging.**

In this paper, we present an effective testing framework for decision-making policies. The framework adopts a generative diffusion model-based test case generator that can easily adapt to different search spaces, ensuring the practicality and validity of test cases. Then, we propose a termination state novelty-based guidance to diversify agent behaviors and improve the test effectiveness. Finally, we evaluate the framework on five widely used benchmarks, including autonomous driving, aircraft collision avoidance, and gaming scenarios. The results demonstrate that our approach identifies more diverse and influential failure-triggering test cases compared to current state-of-the-art techniques. Moreover, we employ the detected failure cases to repair the evaluated models, achieving better robustness enhancement compared to the baseline method.

***Index Terms*—generative model, testing, decision-making policies**

I. INTRODUCTION

As artificial intelligence (AI) advances, approaches such as deep learning (DL), reinforcement learning (RL), and imitation learning (IL)-based decision-making policies have been employed to efficiently solve complex MDPs. Although these methods have demonstrated effectiveness in various applications such as gaming [1], [2], Go [3], and robot manipulation [4], they may also exhibit erroneous behaviors leading to potential errors [5]. Such errors could have catastrophic consequences in safety-critical domains, for instance, causing crashes in autonomous vehicles [6] and even jeopardizing human safety [7]–[9]. Given the growing reliance on decision-making policies, there is an urgent and critical need to develop software engineering techniques tailored to ensure the reliability of these policies.

Testing decision-making policies by exploring the scenarios that cause failures during execution is a promising approach to ensuring their reliability. However, since these policies often employ DNN models, they inherit both the benefits and drawbacks of such models, rendering their testing inherently challenging and time-consuming. In addition, detecting failures in decision-making policies of real-world MDPs is particularly challenging due to the infinite number of states. As it is impossible to exhaustively explore all states and actions of decision-making policies [5], potential failures may persist in the unexplored regions of the huge state space. There have been a number of research efforts tailed for testing decision-making policies. Previous testing methods usually rely on traditional software testing techniques such as search-based testing [5], [10]–[14] or metamorphic testing [15]. Especially, search-based techniques mutate the intermediate or initial state of MDP environments to inflate the scale of test cases and reveal failures in decision-making policies. While metamorphic testing techniques explore cases that violate predefined metamorphic relations. However, due to the aforementioned difficulties, developing scalable and effective testing methods for decision-making policies still remains open and challenging.

In this paper, we develop a methodology for decision-making policies, primarily focusing on two key aspects of testing these policies: agent behavior diversity and test effectiveness. For agent behavior (test case) diversity, since executing similar test cases tends to exercise similar parts of the source code (model), relying on diverse test cases could increase the exploration of the fault space and thus increase the detection rates [16]–[18]. Besides, due to the black-box mechanism of DNN-based systems, diversity is a more practical and effective metric to be used as guidance for testing DNN-based systems [19], [20]. Second, test effectiveness is particularly important for complex decision-making systems, as they often require significant resources to operate and repair. This necessitates that test cases should be effective in identifying potential failures efficiently and accurately. Furthermore, effective test cases could detect more corner cases, which thus increases the possibility of detecting severe vulnerabilities in decision-making policies.

This paper presents a scalable and effective testing framework for decision-making policies, focusing on two aforementioned key aspects. First, we employ a generative diffusion model for producing test cases. Specifically, we first train the diffusion model using normal (random) input for decision-making policies, enabling it to capture the distribution of normal test cases. We then introduce novelty-based guidance to measure the freshness of the trajectories, which effectively guides the testing framework and ensures agent behavior diversity. In particular, we diversify the agent behaviors by maximizing the

novelties of the termination states in interaction trajectories [21]. After training the diffusion model, we incorporate the proposed novelty-based guidance with a joint objective function to fine-tune the diffusion model-based test case generators, thereby encouraging the framework to generate valid test cases with diverse behaviors and a higher likelihood of triggering failures.

We evaluate our approach on five widely used MDP solving tasks, encompassing four scenarios: gaming [22], cooperative navigation (Coop Navi) [23], aircraft collision avoidance [24], and autonomous driving [25]. Compared to the state-of-the-art techniques, our method detects more failure cases with a broader range of state coverage. Additionally, we compare our proposed novelty-based guidance with other existing guidance techniques for failure detection. The findings reveal that novelty-based guidance is more effective and contributes to diversifying agent behaviors. Lastly, we utilize the detected failure cases to repair the evaluated models. The results indicate that failure cases identified by our method could better enhance the robustness of decision-making policies, demonstrating the efficacy of our approach.

In summary, this paper makes the following contributions:

- We present a scalable and effective testing framework for decision-making policies. Our framework is designed to diversify agent behaviors and thus effectively test decision-making policies.
- We utilize a generative diffusion model that can adapt to different search spaces, capture the distribution of test cases and effectively generate test cases.
- We propose new novelty-based guidance to measure the freshness of trajectories and diversify agent behaviors, incorporating it with the diffusion model to enable effective testing for decision-making policies.
- We conduct an extensive evaluation of our proposed methods, comparing them with state-of-the-art baselines. The results highlight the effectiveness of our techniques in improving the robustness of decision-making policies. To facilitate further in-depth research in this field, all materials are made publicly accessible.

The rest of this paper is organized as follows: Section II gives an in-depth explanation of the technical background. Section III is the overview of our method. Section IV introduces the implementation details. Section V presents the results of our evaluation. Section VI provides a detailed analysis of the benefits of using diffusion models and the development of novelty-based metrics in generating test cases for decision-making policies. Section VII discusses threats to the validity of our work. Section VIII introduces related works. Finally, section IX is the concluding remarks.

II. BACKGROUND

A. Markov Decision Processes

A Markov Decision Process (MDP) is a four-tuple $\langle S, A, R, P \rangle$, where S and A represent the sets of states and actions. $R(\cdot)$ and $P(\cdot)$ denote the reward function and state transition function, respectively. In this framework, an agent interacts with the environment at each time step t by observing the current state $s_t \in S$, selecting an action $a_t \in A$ to execute, receiving an immediate reward $r_t = R_a(s_t, s_{t+1})$ after performing a_t, and then transitioning to a new state $s_{t+1} \sim P(s_t, a_t)$. The $R(\cdot)$ specifies the reward the agent receives for performing a specific action in a particular state. At the same time, the $P(\cdot)$ describes the probability distribution over possible following states given the current state and action.

In an MDP environment, the agent selects an action to execute based on a policy $\pi(\cdot)$, represented as $a \sim \pi(s)$, and interacts with the environment, generating a trajectory denoted as $[(s_0, a_0, r_0), ..., (s_t, a_t, r_t), ...]$ where the subscripts indicate different time steps. The return of each trajectory is defined as the sum of discounted rewards received up to time step T, calculated as $\sum_{t=0}^{T} \gamma^t r_t$, where $\gamma \in [0, 1)$ is a discount factor applied to future rewards. Solving MDPs, such as reinforcement learning (RL), generally involves finding an optimal policy that maximizes the expected returns. This is achieved by iteratively improving the policy through trial-and-error interactions with the environment to find the optimal policy that yields the highest expected return over the long term.

B. Decision-Making Policies

Several types of decision-making policies can be used for solving MDPs, including Deep Neural Networks (DNNs), Deep Reinforcement Learning (DRL), Multi-Agent Reinforcement Learning (MARL), and Imitation Learning (IL). We introduce the above decision-making policies as follows:

- **Deep Neural Networks (DNNs)**: DNNs can represent the value or policy function in an MDP. DNNs can learn complex state space representations, allowing for more accurate and efficient policy or value estimation.
- **Deep Reinforcement Learning (DRL)**: DRL is a combination of reinforcement learning and deep neural networks [26]. DRL algorithms, such as Deep Q-Networks (DQN) [1], can learn policies directly from raw sensory input without using hand-crafted features.
- **Multi-Agent Reinforcement Learning (MARL)**: MARL involves multiple agents that interact with each other and the environment [27]. MARL algorithms, such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [23], can learn coordinated policies for multiple agents in a shared environment.
- **Imitation Learning (IL)**: IL involves learning policies by imitating the behavior of an expert [28]. IL algorithms, such as Behavioral Cloning (BC) [29] and Inverse Reinforcement Learning (IRL) [30], can learn policies from demonstrations or expert feedback, which can be more efficient and reliable than trial-and-error exploration.

C. Guidance for Testing Decision-Making Policies

Guidance in testing systems is essential for ensuring the underlying components are thoroughly and efficiently tested. Previous works utilize various guidance to improve the test effectiveness of decision-making policies. In addition, such

guidance effectively measures the test cases by the probability of triggering failures. We introduce the following three guidances used in the experiments.

- **State coverage-based guidance:** Existing works demonstrate that covering more states benefits failure detection. For instance, Pang *et al.* proposed density in *MDPFuzz*, a state coverage measurement of trajectories [12]. A low density indicates that the current trajectory has more new patterns not yet explored. When the densities exceed a threshold, the test cases will be mutated in *MDPFuzz*.
- **Robustness-based guidance:** Previous works state that failures of decision-making policies might locate in scenarios where the agent performance is not robust. For example, sensitivity is discussed by previous works as a robustness measurement [12], [15]. Sensitivity represents the deviation of trajectory rewards before and after implementing a slight mutation of the initial states. A higher sensitivity implies a less robust policy for the test cases.
- **Reward-based guidance:** Previous works also discussed that the reward, as an essential component in MDP, is an effective indicator of triggering failures. The failure cases might occur near the low-reward states and trajectories [5], [13].

In this paper, we propose a termination state-novelty-based guidance to diversify the agent behaviors and explore more failures of the decision-making policies.

D. Diffusion Model

Diffusion models are likelihood-based probabilistic generative models. The original idea comes from cognitive psychology and is adapted for modeling complex data distribution such as images, audio, or text [31]. Compared with previous generative models such as generative adversarial network (GAN) and variational auto-encoder (VAE), diffusion models offer superior convergence in distribution modeling due to the autoregressive-based modeling process [32]. The diffusion-based generative models assume that the data is generated by a Markov process that involves iteratively removing a Gaussian noise from a random noise signal.

Formulation. Diffusion models aim to model the distribution $p_\theta(\mathbf{x}_0)$ of a dataset of interest, which consists of forward and backward processes. In particular, during the forward noising process q, an encoding process that gradually transforms the data distribution into a standard Gaussian $N(0, I)$. The models then use a learnable denoising function p_θ, a decoding process that reverses the transformation. Once the denoising function is learned, new samples from the data distribution can be generated by iteratively applying the reverse denoising steps $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ after sampling from the standard Gaussian. As the latent distribution of x_t is nearly a standard Gaussian distribution, we manage to learn the reverse distribution $q(x_{t-1}|x_t)$ acquire a sample from $q(x_0)$, generating a novel data point from the original data distribution. Figure 1 shows the process of training diffusion models, which mainly includes the encoding and decoding procedures.

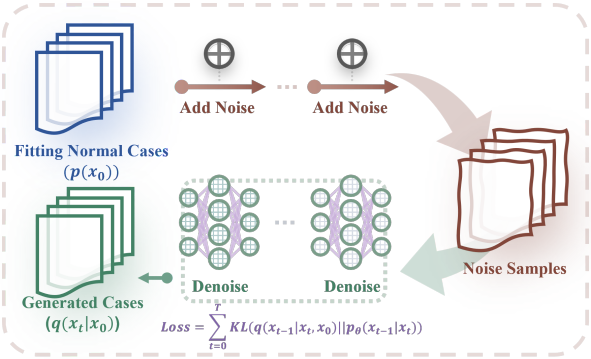

Fig. 1: Brief illustration of the training process of diffusion models.

Objective. The diffusion model is trained to maximize the marginal log-likelihood, $\log p_\theta(\mathbf{x}_0)$. However, since it is hard to optimize the marginal log-likelihood directly, similar to VAE optimization, we instead minimize the expectation of variant lower bounds, L_{vlb}, in the backward process, as shown in Eq. 1.

$$L_{\text{vlb}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \geq -\log p_\theta(\mathbf{x}_0) \quad (1)$$

To train the models, the training objective can be transformed equivalently as minimizing the negative L_{vlb}:

$$\theta^* = \text{argmin}_\theta L_{\text{vlb}} \quad (2)$$

III. METHODOLOGY

Figure 2 illustrates the workflow of our method, which is composed of three modules: (1) a *Normal Test Case Generator* that generates random test cases for the environment, (2) an *Execution Module* that takes test cases as input, produces interaction trajectories, and calculates metrics for further analysis, and (3) a *Diffusion Model-Based Test Case Generator* that creates test cases with higher probabilities of causing failures. Our method consists of two stages. In the first stage, the diffusion model captures the distribution of test cases by training on test cases generated from the *Normal Test Case Generator* with Eq. 1 as loss function. In the second stage, the diffusion model is fine-tuned using both normal and generated test cases from the *Execution Module*. Those cases are employed to detect failures in decision-making policies. Additionally, we fine-tune the diffusion models to learn the desired test case distribution using our proposed loss (Eq. 6) which takes the novelties of both normal and generated test cases into account.

A. Novelty-based Guidance

Previous studies have shown that generating test cases covering fresh states and diversifying agent behaviors can effectively detect failures [33]. Therefore, methods have been proposed to measure state freshness, such as density-based measurement [12], generalizable novelty measurement [34], distance-based novelty [35], and topological similarity measurement [36], [37]. However, recent studies suggest that measuring

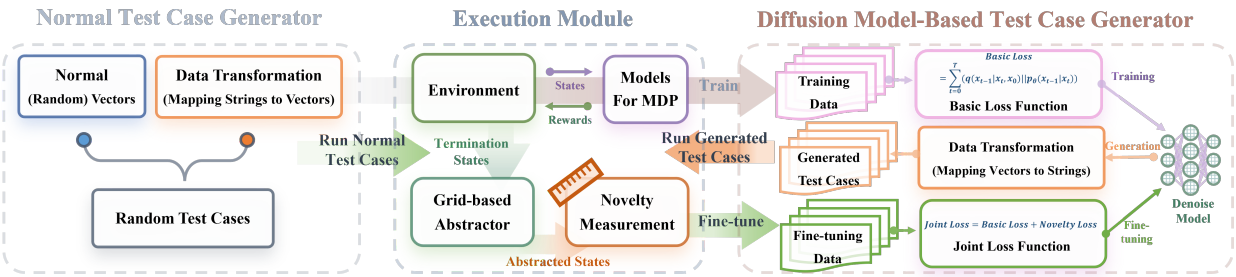

Fig. 2: The overview of our method.

topological similarities can more effectively cluster similar concrete states in MDP environments [36], [38]. Therefore, in this work, we adopt a grid-based clustering method to measure the topological similarities of states [36], [37], [39]. Additionally, unlike previous works that measure the novelty of entire trajectories [12], we directly measure the novelty of the termination state, which is the last concrete state of each trajectory. The reason is that diversifying the termination states can effectively benefit the exploration of the state spaces [21].

Given a K-dimensional state space R^K, the termination state s is represented as $(s^0, ..., s^{K-1})$. The value of the i-th dimension can be scaled into $[l_i, u_i]$, where the l_i and u_i are the lower and upper boundaries. We equally split the value scopes into N intervals of each dimension. In this way, we convert the continuous state space into a discrete state space with finite grids on each dimension as follows:

$$e_n^i = [l_i + n \times \frac{u_i - l_i}{N}, l_i + (n+1) \times \frac{u_i - l_i}{N}] \quad (3)$$

where different concrete termination states s, which locate in the same grid, will be assigned by a common and unique label $\hat{s}$ as follows:

$$\hat{s} = \{s | s^i \in e_n^i, n \in [0, N-1], i \in [0, K-1]\} \quad (4)$$

We apply grid-based clustering to transform the concrete termination states into countable abstract states. Topologically similar concrete states will fall into the same grid as an abstract state. Each abstract state represents a cluster. We count the occurrences concrete states in $\hat{s}$ by $\eta_{\hat{s}}$. Greater occurrences indicate less novelty of the concrete states. During the testing, we calculate the novelties of abstract states $\rho_{\hat{s}}$ based on the occurrences:

$$\eta_{\hat{s}} = |\{s | s \in \hat{s}\}|, \rho_{\hat{s}} = \frac{1}{\exp{(\eta_{\hat{s}} - 1)}} \quad (5)$$

where the abstract states with greater occurrences $\eta_{\hat{s}}$ have lower novelty $\rho_{\hat{s}}$. During the testing phase, we utilize a memory module in the form of a key-value store to store the abstract states and their corresponding occurrences. This straightforward structure allows for efficient storage and retrieval of states, as demonstrated in previous works [36], [38].

An example of a grid-based state cluster in a 2-dimensional state space is shown in Figure 3. The figure demonstrates that several concrete states (i.e., points on the left) are clustered into three types of abstract states (i.e., clusters): abstract states *4, 5, 9, 17, 20* with occurrences $\eta = 1$, abstract state *11* with occurrences $\eta = 2$, and abstract state *19* with occurrences $\eta = 3$. Based on the computation of novelties, their novelties are 1, $\frac{1}{e}$, and $\frac{1}{e^2}$, respectively. Hence, an abstract state with higher occurrence has a lower novelty. It should be noted that our experiments involve agents operating in higher-dimensional state spaces.

B. Test Case Generator

This work introduces an innovative approach of utilizing a diffusion model as a test case generator, illustrated on the right side of Figure 2. The test case generator follows an *encoding-decoding* workflow.

In the *encoding* stage, we introduce standard Gaussian random noise to the normal test cases. Note that normal test cases refer to the test cases randomly generated using the default method of the environment in which the test case components are within a valid range. The noise is gradually refined through multiple diffusion steps until a standard Gaussian distribution closely approximates the distribution of the normal cases. In the *decoding* stage, we use a denoising function p_θ in the form of a fully connected feed-forward neural network to recover the encoded samples (noise) back to normal cases. Diffusion model-based test case generators learn the correspondence and transformation between standard Gaussian noisy samples and normal cases. During the generation, we sample a random noise the same size as the test cases and use the denoising function p_θ to recover the noise to a new test case.

To learn the denoising function p_θ, we update θ, the parameters of the diffusion model, to minimize the gap between the latent distribution of normal cases and generated cases. However, such a process can only guarantee that the generated test cases are similar to the normal cases, which is insufficient to detect more failures. We integrate a novelty-based measurement as part of the loss term to tackle this pitfall. Specifically, we encourage the diffusion model to fit test cases that induce the termination states with high novelties (*i.e.*, low novelty loss). On the contrary, we force the diffusion model to update from the status that generates test cases inducing low-novelty

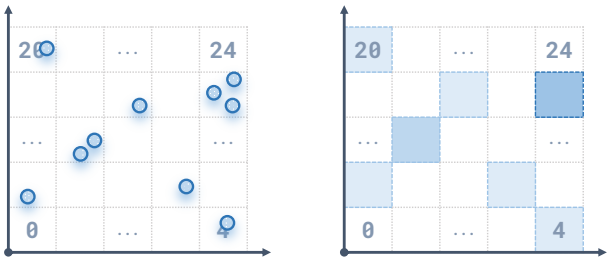

Fig. 3: The process of grid-based abstraction in 2-dimensional state space can be illustrated with a schematic diagram. The diagram on the left side depicts the specific termination states, while the right side represents the abstract states generated by grid-based clustering.

termination states. The final loss then be optimized as follows:

$$\begin{aligned}\theta^* &= \text{argmin}_\theta L_{\text{vlb}} + L_{\text{novelty}} \\ &= \text{argmin}_\theta L_{\text{vlb}} + \lambda(1-\rho)\end{aligned} \tag{6}$$

where the L_{vlb} is the basic loss of diffusion models, the L_{novelty} is used to compute the loss based on abstract state novelties. The λ is a factor to control the value magnitude of L_{novelty}. This way, we can update the θ^* to fit the sample with high novelties. In addition, add the novelty loss $1-\rho$ to the objective function L_{vlb} instead of the only loss in each step (*i.e.*, L_t), making the training process more stable.

Moreover, we implement a simple *Data Transformation Module* to convert the text-based states into float numbers. For example, CARLA uses text-based weather settings, such as *WetCloudyNoon* and *SoftRainSunset*. These components cannot be generated by general diffusion models which output float vectors. Therefore, we use a float number that scales in [0,1] to represent the weather. We split the value scope [0,1] to equal intervals, and each interval corresponds to a weather text. With the *Data Transformation Module*, the diffusion model can quickly adapt to heterogeneous test case search space.

C. Testing Framework

The test framework works in two stages. In the first stage, we randomly sample normal initial states of the environment as test cases using the *Normal Test Case Generator*. We then use the collected normal test cases to train the *Diffusion model-based Test Case Generator*. The diffusion model captures the distribution of normal test cases, ensuring the validity of generated test cases. However, we do not execute the normal test cases in the first stage, and novelty-based guidance metrics are not calculated or involved in the training process of the diffusion model. This ensures that the training is stable in the early stage and that the diffusion model can fit the normal case distribution without novelty-based guidance.

The second stage of the test process involves three steps: collecting test cases from the *Normal Test Case Generator*, executing and evaluating the test cases using novelty-based metrics in the *Execution Module*, and fine-tuning the diffusion model based on the collected and generated test cases and the metrics. We use the *Grid-based Abstract* module to compute the novelty-based metrics. During the fine-tuning process, we execute the test cases generated by the diffusion model to detect failures and update the model according to the joint optimization in Equation 6. This fine-tuning and testing process is repeated until the testing is complete.

Algorithm 1 Testing of Decision-Making Policies.

Input: $\mathcal{P}$: Decision-Making Policy, $\mathcal{E}$: MDP Environment, $\mathcal{D}$: Diffusion Model, $\mathcal{M}$: Metrics Memory (Novelty)
Output: $\mathcal{F}$: Failures detected by our method.

```
1: Initialize: Normal_cases : C ← ∅, Generated_cases : G ← ∅, Failure_cases : F ← ∅
2: function NORMAL_CASES_SAMPLE(steps)
3:     while i < Initilzation_steps do
4:         c_i ← Random_inputs()
5:         s_0 ← E.reset(c_i)
6:         S, A, R, CRASH, ... ← E.execution(s_0, P)
7:         i ← i + 1
8:         C.append(c_i), M.update(S, A, R)
9:     return C, M
10: C, M ← NORMAL_CASES_SAMPLE(Sample_steps)
11: D.train(C, Null, Training_steps)
12: while i < Testing_steps do
13:     if i is Tuning_intervals then
14:         C, M ← NORMAL_CASES_SAMPLE(Sample_steps)
15:         D.train(C + G, M, Tuning_steps)
16:         C ← ∅, G ← ∅, M ← ∅
17:     g_i ← D.generate()
18:     s_0 ← E.reset(g_i)
19:     S, A, R, CRASH, ... ← E.execution(s_0, P)
20:     G.append(g_i), M.update(S, A, R)
21:     if CRASH is true then
22:         F.append(g_i)
23:     i ← i + 1
24: return F
```

We implement our method as introduced in Algorithm 1. We use a function named *NORMAL_CASES_SAMPLE* in line 2, which samples normal (random) inputs using $Sample\_steps$. In line 11, the diffusion model $\mathcal{D}$ is trained by the collected normal inputs novelty-based guidance. The reason is that $\mathcal{D}$ captures the distribution of normal cases in the first stage, and a joint optimization with metrics might influence the convergence of $\mathcal{D}$. While testing the policy $\mathcal{P}$, we generate test cases $\mathcal{G}$ by $\mathcal{D}$ in line 17 and execute them in the environment $\mathcal{E}$. The failures are stored in the buffer $\mathcal{F}$. Moreover, we fine-tune the $\mathcal{D}$ at each $Tuning\_intervals$ in line 15. Note that the $Tuning\_steps$ is significantly smaller than the $Training\_steps$, as we need to avoid overfitting the diffusion model $\mathcal{D}$. The metrics computation $\mathcal{M}$ includes the grid-based novelty measurement.

IV. IMPLEMENTATION

This section introduces the implementation details, including the baseline method, the evaluation environment, our hyperparameter settings, and the hardware condition.

A. Baseline Method

We select *MDPFuzz* [12] as the baseline for our comparison. *MDPFuzz* aims to identify failures in decision-making policies

TABLE I: The number of abstract clusters covered, failure clusters detected, failure rates, and diversity rates achieved by the baseline and our method in five tasks.

Tasks	Methods	States	State Clusters	State Diversity	Failures	Failure Rates	Failure Clusters	Failure Diversity
RL_CARLA	MDPFuzz	3,622	197	5.43%	89	2.43%	19	21.34%
	Ours	2,283	260	**11.38%**	163	**7.13%**	41	**25.15%**
IL_CARLA	MDPFuzz	3,171	257	8.10%	118	3.72%	24	20.33%
	Ours	2,401	339	**14.11%**	184	**7.66%**	73	**39.67%**
RL_BipedalWalker	MDPFuzz	4,188	231	5.51%	718	17.14%	48	6.68%
	Ours	5,651	355	**6.28%**	1,181	**20.89%**	114	**9.52%**
DNN_ACAS_Xu	MDPFuzz	21,109	214	1.01%	17	0.08%	9	**52.91%**
	Ours	15,854	209	**1.31%**	48	**0.30%**	18	37.50%
MARL_Coop_Navi	MDPFuzz	258,971	9339	3.60%	32	0.01%	8	25.00%
	Ours	166,869	6682	**4.00%**	104	**0.06%**	80	**76.92%**

that solve MDPs. *MDPFuzz* generates initial states as test cases that can lead to failure-triggering trajectories. To increase its effectiveness, the authors propose a mutation-based approach to enrich the number of test case candidates and a fuzz-based framework to search for test cases efficiently. In addition, *MDPFuzz* uses a state coverage-based guidance named density to diversify the trajectories. As pointed out by Pang *et al.* [12], altering intermediate states in trajectories to detect failures can disrupt the continuity of adjacent states, as trajectories are composed of states with interdependencies [26]. Thus, our proposed methods seek to identify practical failures by utilizing only the initial states of the environments as test cases for testing decision-making policies, which makes *MDPFuzz* the closest comparable technique.

B. Environment and Decision-making Policies

We conducted experiments to evaluate our method on five decision-making tasks: BipedalWalker game in the Gym domain [22], Cooperative Navigation (Coop Navi) task [23], ACAS-Xu collision avoidance task [24], and two CARLA autonomous driving tasks [25]. The decision-making policy for BipedalWalker was an agent trained by TQC, a model-free DRL algorithm [40]. For Coop Navi, we used the agent trained by Pang *et al.* [12] using a multi-agent reinforcement learning (MARL) algorithm. In addition, we used a DNN model in the official build as the decision-making policy for ACAS-Xu. Finally, for CARLA autonomous driving tasks, we used a DRL-based agent and an IL-based agent, which were obtained from the CARLA Autonomous Driving Challenge [1] and Leaderboard [2], respectively.

C. Experimental Settings and Hyperparameters

We performed tests on CARLA and BipedalWalker for 12 hours, while the remaining tasks were tested for one hour, with duration depending on their complexity. To ensure fairness, we used the hyperparameters from the official open-source build of *MDPFuzz*. For our method, we used a diffusion model as a test case generator and trained it for 100 epochs using 1,000 normal (random) test cases before testing. During each training epoch, the diffusion model performs 50 add-noise steps (encoding) and 50 denoising steps (decoding). In the testing phase, we fine-tuned the diffusion model iteratively using 100 normal test cases, 50 generated test cases by diffusion model, and their corresponding metrics (such as $L_{novelty}$ in Equation 6) for one epoch. The values of λ in the joint loss function are uniformly set to 0.1. Subsequently, we evaluated the decision-making policies using 10 test cases generated by the diffusion model. The denoising model we used was a fully connected neural network consisting of three hidden layers with a size of [256, 512, 256], and ReLU was used as the activation function.

D. Implementation and Hardware:

Our method is implemented with approximately 2K LOC in Python (ver. 3.7.4). We use Numpy (ver. 1.19.5), and PyTorch (ver. 1.13) to implement and train the diffusion model. The experiments were conducted on high-performance servers equipped with a 14-core Intel i9-10940X CPU, an Nvidia A6000 GPU, and 128GB of RAM, running Ubuntu 20.04. More implementation details could be found in our publicly available code repository [3].

V. EVALUATION

In this section, we present the evaluation to study the effectiveness of our method. We compare our method with *MDPFuzz* [12], a state-of-the-art testing technique for decision-making policies. The evaluation results aim to answer the following research questions (RQs):

- **RQ1:** How many failures can the baseline and our method detect in the selected tasks?
- **RQ2:** To what extent does novelty-based guidance outperform other widely used guidance for detecting failures in decision-making policies?
- **RQ3:** How about the diversity of the covered states and detected failures by the baseline and our method?
- **RQ4:** How much robustness is improved when repairing decision-making policies by the detected failures of the baseline and our method?

[1] https://carlachallenge.org

[2] https://leaderboard.carla.org/leaderboard

[3] https://github.com/lizhuo-1994/mdp_testing

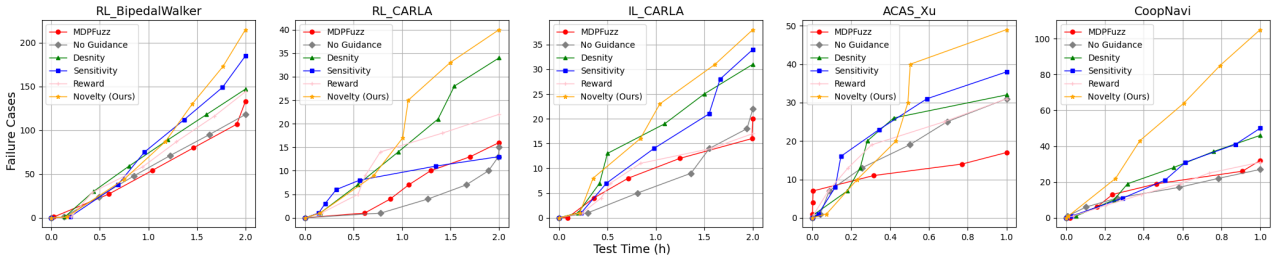

Fig. 4: Detected failures by our method with different guidance on five tasks.

A. RQ1: The Effectiveness of the Baseline and Our Method in Detecting Failures.

We compare the test effectiveness of the baseline and our method by measuring the number of detected failures and the failure rate. The failure rate is calculated as follows:

$$\text{Failure rate} = \frac{|\{s_*|s_* \in S_*\}|}{|\{s|s \in S\}|} \tag{7}$$

where the S represents the group of tested cases, and S_* is the group of the test cases that trigger the failures. A higher failure rate indicates a better test effectiveness of the method. The numbers of detected test cases, detected failures, and failure rates of our method and *MDPFuzz* on the five selected tasks are presented in Table I.

The experimental results demonstrate that our method detects more failures than *MDPFuzz* in all five tasks within the respective testing time budget. Moreover, our method achieves a higher failure rate than *MDPFuzz*. Thus, our method outperforms *MDPFuzz* in detecting failures on all the tasks given the limited time and number of test cases, proving our method's test effectiveness. It should be noted that the results of *MDPFuzz* in this study may slightly differ from the original publication [12] due to variations in hardware and random seeds used in the experiments.

Answer to RQ1: *The experimental results indicate that our method is more efficient in detecting failures than the baseline method within the given time budget. Additionally, the testing results from our method show higher failure rates, indicating that it detects more failures using fewer test cases. In summary, our method is more effective for identifying failures in decision-making policies than the baseline.*

B. RQ2: Comparison between Novelty-based and Other Guidances in Detecting Failures for Decision-making Policies.

To further demonstrate the effectiveness of novelty-based guidance, we compare it with three other widely used guidance (as introduced in II-C) for detecting failures, including density, sensitivity, and reward. Furthermore, we compare test results obtained with and without guidance to train diffusion model-based test case generators. We obtain the official implementation of the aforementioned metrics from the official repository of *MDPFuzz* [12]. All the values of the metrics are normalized into [0, 1] (*i.e.*, the same value scope of novelties) and multiplied by the uniform hyperparameter λ in Equation 6 for the joint optimization of the diffusion model-based test case generators.

Figure 4 shows the results of testing the respective decision-making policies on five tasks by selected guidance. We can see that although the pure diffusion model-based test case generator without specific guidance can produce failure-inducing test cases, the number of detected failures is significantly lower than that of other methods in all five tasks. Furthermore, after integrating the selected guidance into the joint loss function of the diffusion models, our method can identify more failures. Another observation is that, with novelty-based guidance, our method detected the most failures in all five tasks. These results validate that novelty is the most effective guidance for failure detection in decision-making policies.

To further demonstrate the effectiveness of novelty-based guidance in test generation, we replaced the density-based feedback signals of the fuzzing framework in *MDPFuzz* with novelty. As shown in Figure 5, the evaluation results reveal an improvement in detected failures with novelty-based guidance for IL_CARLA, while RL_BipedalWalker experiences a minor increase. The findings above indicate that *MDPFuzz* with novelty-based guidance can identify more failures than the original *MDPFuzz* (with density-based guidance), further emphasizing the effectiveness of novelty in detecting failures of decision-making policies.

Answer to RQ2: *By optimizing the diffusion model-based test case generator with novelty-based guidance, our method achieves the highest number of detected failures among several widely used guidance for testing decision-making policies. Furthermore, incorporating novelty-based guidance improves the test efficiency of the baseline method, demonstrating its effectiveness in detecting failures compared to other forms of guidance.*

C. RQ3: The Diversity of Covered States and Failures Detected by the Baseline and Our Method.

Prior research has demonstrated that diversifying agent behaviors and increasing state space coverage can improve the effectiveness of failure detection [12], [13]. In this section, we analyze the diversity of the baseline and our method by assessing the covered states and detected failures. To achieve this, we utilize grid-based clustering (as illustrated in III-A) to measure the topological similarities of the termination states of trajectories. This approach is employed because grid-based

clustering groups similar concrete states with closer distances in the state space into the same cluster, *i.e.*, the abstract state, thereby increasing the effectiveness of measuring the diversity of agent behaviors.

We evaluate the diversity of agent behaviors using four metrics: the number of covered abstract clusters during the testing, the state diversity rate, the number of failure clusters, and the failure diversity rate. We first obtain a set of covered termination states denoted by S and the corresponding set of clusters $\hat{S}$ to calculate the state diversity rate. Then, we calculate the number of clusters that are covered given the number of concrete termination states (which is equal to the number of test cases and trajectories) using the following formula:

$$\text{State diversity} = \frac{|\{\hat{s}|\hat{s} \in \hat{S}\}|}{|\{s|s \in S\}|} \tag{8}$$

We also measure the failure diversity rate. First, we obtain a set of concrete states S_* and the corresponding set of clusters $\hat{S}_*$ covered by all the failure-triggering trajectories. Then, we calculate the failure diversity rate using the following formula:

$$\text{Failure diversity} = \frac{|\{\hat{s}_*|\hat{s}_* \in \hat{S}_*\}|}{|\{s_*|s_* \in S_*\}|} \tag{9}$$

As shown in Table I, our method covers more abstract clusters of the termination states covered by the test cases than *MDPFuzz* in four tasks. In addition, the number of covered abstract clusters is close to the *MDPFuzz* in DNN_ACAS_Xu, since we cost significantly fewer test cases. Furthermore, our method has achieved higher state diversity rates than *MDPFuzz* in all five tasks. This indicates that our method can cover a broader range of states in the given testing time and the number of test cases than *MDPFuzz*.

The failures detected by our method cover more abstract clusters than *MDPFuzz* in all five tasks. Based on the above results, our method has exhibited higher failure diversity rates than *MDPFuzz* in four tasks, further highlighting its superior performance in diversifying the agent behaviors. Note that our method shows a lower failure diversity in MARL_Coop_Navi. The reason is that *MDPFuzz* detects 32 failures, significantly smaller than ours (*i.e.*, 104), making the failure diversity value greater. Overall, our method has successfully achieved two critical goals: covering a broader and more diverse range of states and detecting a wider range of failures than *MDPFuzz*.

Answer to RQ3: *Our method excels the baseline method in covering a broader range of states and failures within the given number of test cases. These experimental results signify that our method can better diversify the agent behaviors and detect failures in decision-making policies than the baseline method.*

D. RQ4: Robustness Improvement through the Repair of Decision-making Policies using Our Detected Failures.

To improve robustness, we focused on repairing the decision-making policies in RL_BipedalWalker and DNN_ACAS_Xu using the failure trajectories detected by both *MDPFuzz* and our method. We chose these tasks because other decision-making policies require considerable computation resources or are black-box models. Our repair approach fine-tunes the model for 60 epochs in DNN_ACAS_Xu, the same as the repair in *MDPFuzz*, and 100,000 total steps in RL_BipedalWalker. Finally, we evaluated the robustness improvement achieved by our method.

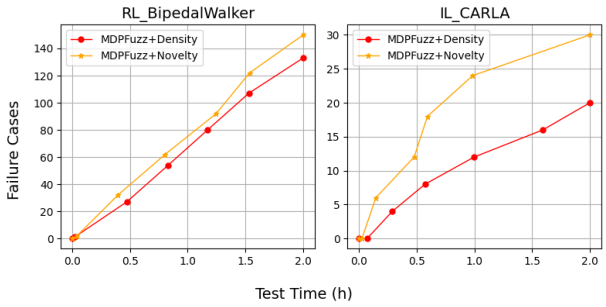

Fig. 5: The testing results of *MDPFuzz* using novelty-based guidance.

Table II presents the results of repairing the decision-making policies of RL_BipedalWalker and DNN_ACAS_Xu using the detected failure trajectories of *MDPFuzz* and our method. For the repair process, we fine-tuned the model in 60 epochs for DNN_ACAS_Xu, the same as the repair in *MDPFuzz*, and 100,000 total steps for RL_BipedalWalker. We randomly shuffled the test cases generated by the baseline and our method and trained the decision-making policies on the trajectories produced by the generated test cases. Moreover, we add the same amount of normal test cases for repairing to avoid overfitting. The results show that policies repaired by our detected failure trajectories have 477 and 8 failures in RL_BipedalWalker and DNN_ACAS_Xu, respectively, while policies repaired by *MDPFuzz* detected 629 and 14 failures, respectively. Hence, we conclude that our detected failures are more effective in improving robustness than *MDPFuzz*.

Answer to RQ4: *The failure cases detected by our method can help improve the robustness of decision-making policies. Additionally, repairing evaluated models using our detected failure cases leads to a more significant improvement in robustness compared to the baseline method.*

VI. DISCUSSION

A. Potential for Adopting Generative AI in Other Software Engineering Areas

Our experiment demonstrates that integrating generative models with novelty-based guidance significantly enhances both agent behavior diversity and test effectiveness. Additionally, we discovered that employing generative AI for test case generation can also help improve the validity of test cases, which is a crucial factor to consider when developing testing frameworks. As discussed in [41], [42], the absence of test case validation could substantially inflate coverage and escalate test costs,

TABLE II: The number of detected failures before and after repair.

Item	RL_BipedalWalker	DNN_ACAS_Xu
MDPFuzz	718	17
MDPFuzz (Repair)	629	14
Ours	1,181	48
Ours (Repair)	477	8

leading to diminished test effectiveness. Generative models facilitate flexible adaptation to various search spaces with minimal effort. Moreover, they can dependably validate input by capturing the standard case distribution and generating valid and novel cases without imposing additional constraints.

These findings may have broader implications, suggesting that we should contemplate utilizing these generative models and adapting them for other software engineering tasks. For instance, the diffusion model used in generation tasks can be designed to generate valid data samples by explicitly constraining the diffusion process. In particular, the diffusion process can be designed to prevent the generation of invalid samples, such as images with missing pixels or audio with random noise. One way to do this is by using a diffusion process that respects the geometry or structure of the data. For example, in image generation, the diffusion process can be designed to respect the spatial structure of the pixels so that each pixel is generated based on its neighbors, which can help prevent the generation of invalid images with missing pixels or artifacts. Another way to ensure the generation of valid samples is to use a diffusion process that is conditioned on the input data or a specific task. This can help guide the diffusion process toward generating valid samples consistent with the input data or the desired output.

B. Detailed Design Philosophy of Our Testing Framework

Our method goes beyond the goal of detecting more failures than the baseline on selected tasks. The reason is that testing decision-making policies is challenging due to the massive state space of complex environments, the high cost of test execution, and the black-box nature of DNN-based components. As a result, conducting exhaustive testing of decision-making policies is not feasible [5]. Moreover, failures in decision-making policies tend to be concentrated on similar scenarios, which can negatively impact testing effectiveness. Instead, recent studies have shown that generating test cases that induce diverse behaviors can improve the testing of decision-making policies [12], [13]. In this study, we also aim to detect a more diverse range of failures, and therefore, we utilize novelty-based guidance to enhance the test case generation. The results indicate that our method's detected failures can better enhance the policy's robustness. Thus, an effective testing method for decision-making policies should focus on detecting a broader range of failures and exploring a wider range of agent behaviors.

Note that although we have many choices in measuring the freshness of the termination states, such as *KNN* [38], in this paper, we select grid-based abstraction to discretize the concrete termination states and adopt the occurrence of the abstract states as novelty measurements. The reason is that grid-based abstraction is more efficient and effective in grouping topologically similar states [36]. However, we found that the abstraction granularity (grid numbers) on each dimension can affect the correctness of abstraction. For example, a smaller grid number makes more concrete states fall into the same grid and causes the abstraction to be less representative. On the contrary, a greater grid number increases the number of grids, thus harming the generality. In our method, we set the grid number to 5, 5, 5, 5, and 2 in RL_CALRA, IL_CARLA, RL_BipelWalker, DNN_ACAS_Xu, and MARL_Coop_Navi, respectively.

The computation of novelties in Equation 5 involves the reciprocal of the exponentiated value of abstract state occurrences η. This leads to novelty values ρ that change more rapidly when η is relatively small, making it easier to identify new abstract states. To illustrate this, consider the abstract states in Figure 3, where abstract states 4, 11, and 19 occurred 1, 2, and 3 times, respectively. Their corresponding novelties are 1, $\frac{1}{e}$, and $\frac{1}{e^2}$, respectively. The novelties decrease more quickly when state occurrences are relatively small, as $(1 - \frac{1}{e}) > (\frac{1}{e} - \frac{1}{e^2})$. This computation facilitates a clearer differentiation between new and covered states, which is advantageous for optimizing the diffusion model to discover and explore more novel states.

Both our method and the *MDPFuzz* approach have demonstrated the ability to detect a wider range of failures in decision-making policies through diversification of termination states and entire trajectories, respectively. However, measuring the novelty of trajectories (*i.e.*, density in *MDPFuzz*) can be computationally complex and inefficient, as they are typically high-dimensional vectors, as mentioned in *MDPFuzz*. Instead, we propose measuring novelty based on termination states, which is more computationally efficient. Moreover, we suggest that measuring the novelty of multi-step states (patterns) could be a more effective approach for identifying novel trajectories [36]. In addition, we believe that using a generalizable memory [43] such as DNNs to store and fit the covered states or patterns would be more scalable than the key-value memory used in our method. This is because a generalizable memory can more easily adapt to higher-dimensional states.

Overall, the primary objective of our method is to develop an effective testing framework for decision-making policies. Our focus is not only on detecting more failures but also on identifying more diverse trajectories. Furthermore, our approach can be applied to new testing tasks without requiring significant engineering effort.

VII. THREATS TO VALIDITY

A. Internal Threats

An internal threat to our method's validity is the diffusion model's fine-tuning process. During fine-tuning, the diffusion model is trained using the normal cases from the initialization, which could potentially harm the model's convergence. To mitigate this issue, we adopt two solutions: (1) we introduce a factor to control the value range of the novelties during joint

optimization, and (2) we monitor the loss of the diffusion model to prevent overfitting. Although we did not observe any invalid test cases generated by our method, this threat still exists and warrants further investigation, particularly in scenarios involving longer-term fine-tuning of the diffusion models.

B. External Threats

The fact that we selected only a limited number of decision-making and repair tasks could be an external threat to the generalization of our method. Despite the limitations in computational resources, our method demonstrated its effectiveness. However, further experiments on a broader range of tasks are necessary to confirm the robustness and versatility of our method, particularly in safety-critical and real-world scenarios. We aim to continuously enhance our method for more tasks and deeper insights. We leave the above topics as future works.

VIII. Related Work

A. DNN Testing

DNN testing is an essential quality assurance for deep learning-based models and applications [44]. The related works have drawn much attention in both software engineering and artificial intelligence communities in recent years, including the tests of correctness [45], [46], relevance [47]–[49], robustness [50]–[52], faireness [53], [54]. Unlike traditional software test techniques, DNN testing usually leverages unique characteristics such as neuron coverage [55]. The above DNN testing methods are applied to various tasks such as autonomous driving [56], [57], machine translation [58], [59], nature language inference [60], [61]. However, testing DNN-based decision-making policies still establishes challenges such as oracle definition and test case generation [12], [15], [62]. This paper mainly focuses on testing decision-making policies for solving MDPs, where the existing testing methods for DNN are not applicable.

B. Decision-making Policy Testing

Previous works have stated that existing decision-making policies require more thorough tests [63]. Most existing works focus on testing decision-making policies by exploring the test case search spaces [5], [10]–[14], [64], which achieved superior testing performance. Alessandro *et al.* applied the search-based testing methods to autonomous driving systems (ADS) [11] with promising results. Koren *et al.* used perturbation-based modification on the elements of the environment to test the collision of vehicles [65]. Lee *et al.* adopted adaptive stress testing (AST) to decision-making policies in MDP environment [66]. Another inspiring work is testing decision-making policies by searching the *inconsistency* via metamorphic relaxations [15]. Besides, a few words about finding bugs in the decision-making policies in hybrid systems have been proposed [67]. Dreossi *et al.* performed tests on cyber-physical systems (CPS) [68], a type of physical MDP environment. Recently, the tests and verification of decision-making policies by formal methods have been proven effective [69], [70].

These methods usually require domain-specific knowledge to design the mutation strategies and metamorphic relations. This paper discusses a novel solution for generating test cases by generative models. Furthermore, existing works on testing decision-making policies still exhibit room for improving test effectiveness and diversity. Our method adopts a diffusion model-based test case generator with novelty-based guidance to mitigate the above challenges.

C. Generative Model-based Testing

DeepRoad, as presented in Zhang *et al.*, utilizes GANs to automatically synthesize large-scale driving scenes and assess the consistency of driving systems [71]. Similarly, Li *et al.* demonstrates the effectiveness of similar methods in generating and identifying critical environmental conditions [72]. BMT [73], proposed by Deng *et al.*, adopts generative models to produce logical metamorphic relations for autonomous driving tasks. In addition, generative model-based test generation has proven effective in performance testing of complex software [74]. Previous studies have shown that generative models effectively generate numerically critical test cases. Furthermore, in the tasks mentioned above, generative model-based generators can produce diverse and valid test cases, as demonstrated by Fontes *et al.* in their integration work [75].

Another application of generative models in testing DNN models by Swaroopa *et al.* shows that existing test case generators [50], [51], [76] for DNN models lack validity since the test cases might be out-of-distribution (OOD) against the training data. Moreover, they proposed a generative model (VAE) -based test case generator to ensure the cases are in the training data distribution [41]. In this paper, we consider that using generative models as test case generators benefit not only test cases' validity but also test implementation efficiency since we can quickly generate test cases by sampling from the learned distribution instead of carefully designing the test case generation constraints [12], [41].

IX. Conclusion

In this work, we propose a novel testing framework for decision-making policies, incorporating generative diffusion models as test case generators and novelty-based guidance to diversify agent behaviors. Our experimental results demonstrate that our method outperforms baseline techniques in detecting a wider range of failures and enhancing policy robustness. We show that diversifying agent behaviors can help effectively test decision-making policies. Moreover, using our detected failure cases to repair the decision-making policies results in a more significant robustness improvement than the baseline method. Our future work involves continuously enhancing our method to test and improve decision-making policies. This will be achieved through the following steps: (1) expanding our techniques to address increasingly complex and safety-critical tasks; (2) investigating novel guidance strategies to discover more corner failures; (3) conducting more effective robustness enhancements for existing decision-making policies.

ACKNOWLEDGMENT

This work was supported in part by JSPS KAKENHI Grant No.JP21K11841, No.JP23K11049 and No.JP23H03372, JST SPRING Grant No.JPMJSP2136, JST Grant No.JPMJFS2132, the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008) as well as Canada CIFAR AI Chairs Program, the Natural Sciences and Engineering Research Council of Canada (NSERC No.RGPIN-2021-02549, No.RGPAS-2021-00034, No.DGECR-2021-00019).

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[4] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 590–595.

[5] A. Zolfagharian, M. Abdellatif, L. Briand, M. Bagherzadeh *et al.*, "Search-based testing approach for deep reinforcement learning agents," *arXiv preprint arXiv:2206.07813*, 2022.

[6] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[7] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[8] Z. Zhu and H. Zhao, "A survey of deep rl and il for autonomous driving policy learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 043–14 065, 2021.

[9] M. R. Endsley, "Autonomous driving systems: A preliminary naturalistic study of the tesla model s," *Journal of Cognitive Engineering and Decision Making*, vol. 11, no. 3, pp. 225–238, 2017.

[10] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2009.

[11] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Generating avoidable collision scenarios for testing autonomous driving systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 375–386.

[12] 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.

[13] M. Tappler, F. C. Córdoba, B. K. Aichernig, and B. Könighofer, "Search-based testing of reinforcement learning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, L. D. Raedt, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 503–510, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2022/72

[14] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Simultaneously searching and solving multiple avoidable collisions for testing autonomous driving systems," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 1055–1063.

[15] H. F. Eniser, T. P. Gros, V. Wüstholz, J. Hoffmann, and M. Christakis, "Metamorphic relations via relaxations: An approach to obtain oracles for action-policy testing," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 52–63.

[16] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, mar 2013. [Online]. Available: https://doi.org/10.1145/2430536.2430540

[17] H. Hemmati, Z. Fang, and M. V. Mantyla, "Prioritizing manual test cases in traditional and rapid release environments," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, 2015, pp. 1–10.

[18] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[19] M. A. Langford and B. H. C. Cheng, "Enki: A diversity-driven approach to test and train robust learning-enabled systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 15, no. 2, may 2021. [Online]. Available: https://doi.org/10.1145/3460959

[20] Z. Aghababaeyan, M. Abdellatif, L. Briand, R. S, and M. Bagherzadeh, "Black-box testing of deep neural networks through test case diversity," *IEEE Transactions on Software Engineering*, pp. 1–26, 2023.

[21] T. Dai, H. Liu, K. Arulkumaran, G. Ren, and A. A. Bharath, "Diversity-based trajectory and goal selection with hindsight experience replay," in *PRICAI 2021: Trends in Artificial Intelligence: 18th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2021, Hanoi, Vietnam, November 8–12, 2021, Proceedings, Part III 18*. Springer, 2021, pp. 32–45.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[23] 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.

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

[25] 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.

[26] R. S. Sutton, "Reinforcement learning: Past, present and future," in *Simulated Evolution and Learning: Second Asia-Pacific Conference on Simulated Evolution and Learning, SEAL'98 Canberra, Australia, November 24–27, 1998 Selected Papers 2*. Springer, 1999, pp. 195–197.

[27] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[28] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[29] F. Torabi, G. Warnell, and P. Stone, "Behavioral cloning from observation," *arXiv preprint arXiv:1805.01954*, 2018.

[30] A. Y. Ng, S. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, vol. 1, 2000, p. 2.

[31] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[32] C. Luo, "Understanding diffusion models: A unified perspective," *arXiv preprint arXiv:2208.11970*, 2022.

[33] M. Boussaa, O. Barais, G. Sunyé, and B. Baudry, "A novelty search approach for automatic test data generation," in *2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing*. IEEE, 2015, pp. 40–43.

[34] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[35] Z.-W. Hong, T.-Y. Shann, S.-Y. Su, Y.-H. Chang, T.-J. Fu, and C.-Y. Lee, "Diversity-driven exploration strategy for deep reinforcement learning," *Advances in neural information processing systems*, vol. 31, 2018.

[36] Z. Li, D. Zhu, Y. Hu, X. Xie, L. Ma, Y. Zheng, Y. Song, Y. Chen, and J. Zhao, "Neural episodic control with state abstraction," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=C2fsSj3ZGiU

[37] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: Model-based quantitative analysis of stateful deep learning systems," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 477–487.

[38] I. Kuznetsov and A. Filchenkov, "Solving continuous control with episodic memory," *arXiv preprint arXiv:2106.08832*, 2021.

[39] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[40] 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.

[41] S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-aware testing of neural networks using generative models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 226–237.

[42] V. Riccio and P. Tonella, "When and why test generators for deep learning produce invalid inputs: an empirical study," *arXiv preprint arXiv:2212.11368*, 2022.

[43] H. Hu, J. Ye, G. Zhu, Z. Ren, and C. Zhang, "Generalizable episodic memory for deep reinforcement learning," *arXiv preprint arXiv:2103.06469*, 2021.

[44] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[45] 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.

[46] M. H. Asyrofi, F. Thung, D. Lo, and L. Jiang, "Crossasr: Efficient differential testing of automatic speech recognition via text-to-speech," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 640–650.

[47] J. Zhang, E. T. Barr, B. Guedj, M. Harman, and J. Shawe-Taylor, "Perturbed model validation: A new framework to validate model relevance," 2019.

[48] R. Werpachowski, A. György, and C. Szepesvári, "Detecting overfitting via adversarial examples," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[49] A. Gossmann, A. Pezeshk, and B. Sahiner, "Test data reuse for evaluation of adaptive machine learning algorithms: over-fitting to a fixed'test'dataset and a potential solution," in *Medical Imaging 2018: Image Perception, Observer Performance, and Technology Assessment*, vol. 10577. SPIE, 2018, pp. 121–132.

[50] 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.

[51] 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.

[52] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[53] M. H. Asyrofi, Z. Yang, I. N. B. Yusuf, H. J. Kang, F. Thung, and D. Lo, "Biasfinder: Metamorphic test generation to uncover bias for sentiment analysis systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 5087–5101, 2021.

[54] A. Albarghouthi, L. D'Antoni, and S. Drews, "Repairing decision-making programs under uncertainty," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 2017, pp. 181–200.

[55] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[56] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," *Advances in neural information processing systems*, vol. 31, 2018.

[57] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*. Springer, 2018, pp. 408–426.

[58] L. Sun and Z. Q. Zhou, "Metamorphic testing for machine translations: Mt4mt," in *2018 25th Australasian Software Engineering Conference (ASWEC)*. IEEE, 2018, pp. 96–100.

[59] W. Zheng, W. Wang, D. Liu, C. Zhang, Q. Zeng, Y. Deng, W. Yang, P. He, and T. Xie, "Testing untestable neural machine translation: An industrial case," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 314–315.

[60] Y. Nie, Y. Wang, and M. Bansal, "Analyzing compositionality-sensitivity of nli models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 6867–6874.

[61] H. Wang, D. Sun, and E. P. Xing, "What if we simply swap the two text fragments? a straightforward yet effective way to test the robustness of methods to confounding signals in nature language inference tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7136–7143.

[62] D. Zhu, J. Chen, W. Shang, X. Zhou, J. Grossklags, and A. E. Hassan, "Deepmemory: model-based memorization analysis of deep neural language models," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1003–1015.

[63] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[64] J. Uesato, A. Kumar, C. Szepesvari, T. Erez, A. Ruderman, K. Anderson, N. Heess, P. Kohli *et al.*, "Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures," *arXiv preprint arXiv:1812.01647*, 2018.

[65] 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.

[66] R. Lee, O. J. Mengshoel, A. Saksena, R. W. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, "Adaptive stress testing: Finding likely failure events with reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 69, pp. 1165–1201, 2020.

[67] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, "A survey of algorithms for black-box safety validation," *J. Artif. Intell. Res.*, vol. 72, pp. 377–428, 2020.

[68] T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh, "Efficient guiding strategies for testing of temporal properties of hybrid systems," in *NASA Formal Methods*, 2015.

[69] T. Akazaki, S. Liu, Y. Yamagata, Y. Duan, and J. Hao, "Falsification of cyber-physical systems using deep reinforcement learning," in *Formal Methods: 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings 22*. Springer, 2018, pp. 456–465.

[70] A. Corso, R. Moss, M. Koren, R. Lee, and M. Kochenderfer, "A survey of algorithms for black-box safety validation of cyber-physical systems," *Journal of Artificial Intelligence Research*, vol. 72, pp. 377–428, 2021.

[71] 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.

[72] Z. Li, M. Pan, T. Zhang, and X. Li, "Testing dnn-based autonomous driving systems under critical environmental conditions," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6471–6482.

[73] Y. Deng, G. Lou, X. Zheng, T. Zhang, M. Kim, H. Liu, C. Wang, and T. Y. Chen, "Bmt: Behavior driven development-based metamorphic testing for autonomous driving models," in *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET)*. IEEE, 2021, pp. 32–36.

[74] A. Sedaghatbaf, M. H. Moghadam, and M. Saadatmand, "Automated performance testing based on active deep learning," in *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 2021, pp. 11–19.

[75] A. Fontes and G. Gay, "The integration of machine learning into automated test generation: A systematic literature review," *arXiv preprint arXiv:2206.10210*, 2022.

[76] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Deepconcolic: Testing and debugging deep neural networks," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 111–114.