4-2024

Curiosity-driven testing for sequential decision-making process

Junda HE
Singapore Management University, jundahe@smu.edu.sg

Zhou YANG
Singapore Management University, zyang@smu.edu.sg

Jieke SHI
Singapore Management University, jiekeshi@smu.edu.sg

Chengran YANG
Singapore Management University, cryang@smu.edu.sg

Kisub KIM
Singapore Management University, kisubkim@smu.edu.sg

See next page for additional authors

Citation

HE, Junda; YANG, Zhou; SHI, Jieke; YANG, Chengran; KIM, Kisub; XU, Bowen; ZHOU, Xin; and David LO. Curiosity-driven testing for sequential decision-making process. (2024). *ICSE '24: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Lisbon, Portugal, April 14-20*. 1-14.
Available at: https://ink.library.smu.edu.sg/sis_research/9258

Author
Junda HE, Zhou YANG, Jieke SHI, Chengran YANG, Kisub KIM, Bowen XU, Xin ZHOU, and David LO

Curiosity-Driven Testing for Sequential Decision-Making Process

Junda He
Singapore Management University
Singapore, Singapore
jundahe@smu.edu.sg

Zhou Yang*
Singapore Management University
Singapore, Singapore
zyang@smu.edu.sg

Jieke Shi
Singapore Management University
Singapore, Singapore
jiekeshi@smu.edu.sg

Chengran Yang
Singapore Management University
Singapore, Singapore
cryang@smu.edu.sg

Kisub Kim
Singapore Management University
Singapore, Singapore
kisubkim@smu.edu.sg

Bowen Xu
North Carolina State University
Raleigh, United State
bxu22@ncsu.edu

Xin Zhou
Singapore Management University
Singapore, Singapore
xinzhou.2020@phdcs.smu.edu.sg

David Lo
Singapore Management University
Singapore, Singapore
davidlo@smu.edu.sg

ABSTRACT

Sequential decision-making processes (SDPs) are fundamental for complex real-world challenges, such as autonomous driving, robotic control, and traffic management. While recent advances in Deep Learning (DL) have led to mature solutions for solving these complex problems, SDMs remain vulnerable to learning unsafe behaviors, posing significant risks in safety-critical applications. However, developing a testing framework for SDMs that can identify a diverse set of crash-triggering scenarios remains an open challenge. To address this, we propose CureFuzz, a novel curiosity-driven black-box fuzz testing approach for SDMs. CureFuzz proposes a curiosity mechanism that allows a fuzzer to effectively explore novel and diverse scenarios, leading to improved detection of crash-triggering scenarios. Additionally, we introduce a multi-objective seed selection technique to balance the exploration of novel scenarios and the generation of crash-triggering scenarios, thereby optimizing the fuzzing process. We evaluate CureFuzz on various SDMs and experimental results demonstrate that CureFuzz outperforms the state-of-the-art method by a substantial margin in the total number of faults and distinct types of crash-triggering scenarios. We also demonstrate that the crash-triggering scenarios found by CureFuzz can repair SDMs, highlighting CureFuzz as a valuable tool for testing SDMs and optimizing their performance.

CCS CONCEPTS

• **Software and its engineering → Process validation**; • **Computing methodologies → Artificial intelligence**.

*Corresponding author.

KEYWORDS

Fuzz Testing, Sequential Decision Making, Deep Learning

ACM Reference Format:
Junda He, Zhou Yang, Jieke Shi, Chengran Yang, Kisub Kim, Bowen Xu, Xin Zhou, and David Lo. 2024. Curiosity-Driven Testing for Sequential Decision-Making Process. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3597503.3639149

1 INTRODUCTION

Sequential decision-making processes (SDPs) involve a series of interrelated decisions, where each decision depends on the outcome of the previous one. SDPs play a critical role in addressing various complex real-world challenges such as autonomous driving [33], robotic control [38], and traffic control [75]. Recent advances in Deep Learning (DL), e.g., Deep Neural Networks (DNN) [62], Deep Reinforcement Learning (DRL) [64], and Imitation Learning (IL) [40], have led to mature solutions for handling these complex sequential decision-making problems. We refer to these solutions as sequential decision-makers (SDMs). In various applications, such as video game playing [20, 29, 64], and aircraft collision avoidance systems [35], these SDMs demonstrate human-comparable or even superior capabilities.

Despite impressive effectiveness, SDMs are susceptible to learning unsafe behaviors during the training process [56]. As SDMs primarily aim to optimize overall performance, they might not adequately prioritize safety concerns. This learning flaw could potentially lead to catastrophic failures in real-world scenarios [56, 84]. The impact of such risks is particularly severe in safety-critical applications, where the actions of SDMs directly affect human lives, and any negligence could result in disastrous consequences. A heartbreaking report[1] reveals that between July 2021 and May 2022, there were 392 crashes involving autonomous vehicles, leading to five serious injuries and six deaths. This highlights the necessity to thoroughly test SDMs to ensure safety in critical scenarios before their deployment.

[1]www.engadget.com/self-driving-car-technology-crash-data

An ideal testing framework for SDMs must be able to produce a diverse set of crash-triggering scenarios [31]. A scenario, in this context, is defined as the set of environmental properties with which an SDM interacts. For example, in autonomous driving, a scenario would include the description of pedestrians, other vehicles, and traffic conditions. The generation of a diverse set of crash-triggering scenarios is crucial for several reasons. Firstly, identifying duplicate (i.e., the opposite of diverse) crash-triggering scenarios wastes computational resources that could have been used to uncover new bugs. Secondly, less diverse scenarios cover smaller input space, thus uncovering fewer corner case crashes.

However, enhancing the diversity of crash-triggering scenarios is indeed challenging, as it necessitates a method to measure the novelty of a scenario. In the context of sequential decision-making problems, the state space is often high-dimensional and continuous. This makes the computation of novelty difficult and leads to high computational costs, commonly known as the "curse of dimensionality" [39]. Thus, researchers are motivated to propose novelty measures for scenarios in the context of testing SDMs. Nonetheless, existing novelty measures have their limitations. For instance, the density-based method [56] exhibits high computational complexity, especially in environments with high complexity. Moreover, the topological similarity measurement proposed by Li et al. [86] can only measure the novelty of the termination states. While typically SDMs make hundreds of interactions with the environment in each run, and the rich information from the intermediate states is buried. Consequently, finding a more efficient and effective novelty measure that helps in generating diverse test cases for SDMs remains a to be an ongoing challenge.

Inspired by the concept of Random Network Distillation (RND) in reinforcement learning [11], we propose a curiosity mechanism and a novel fuzz testing method to address the aforementioned challenge. Our curiosity mechanism calculates the extent of the fuzzer's curiosity in exploring specific scenarios and encourages the fuzzer to prioritize scenarios with higher curiosity. Specifically, when presented with a scenario, our curiosity mechanism will predict the subsequent states within its environment. The discrepancy between our prediction and the actual outcome (prediction error) reflects the level of the fuzzer's curiosity. A scenario with higher curiosity suggests that it is unfamiliar to the fuzzer, as the fuzzer cannot accurately predict it. An essential advantage of our curiosity mechanism is its computational efficiency. The computational complexity of our curiosity mechanism increases linearly while the environmental complexity increases. By leveraging this approach, we can both effectively and efficiently encourage the fuzzer to explore uncharted territories, thereby increasing the diversity of scenarios generated. The computationally inexpensive feature also allows us to successfully apply our method to complex environments, including self-driving systems.

With the novel curiosity mechanism, we propose CureFuzz (**Cur**iosity-driv**e**n **fuzz**er), a curiosity-driven fuzz testing approach for Sequential Decision Making Process. CureFuzz combines two distinct techniques: (1) a curiosity mechanism that measures the novelty of encountered scenarios and encourages the fuzzer to detect novel and diverse scenarios; (2) a multi-objective seed selection technique in fuzzing that estimates the energy of a seed based on its probability of triggering crashes and exploring novel scenarios. CureFuzz then selects the seed for mutation based on its estimated energy to guide the search for crash-triggering scenarios. The former of these two techniques provides a diversity of generated scenarios, and the latter ensures effectiveness in detecting crash-triggering scenarios. By combining them, CureFuzz achieves a balance between effectiveness and diversity in fuzzing testing of the Sequential Decision Making Process, leading to a significant improvement over the state-of-the-art.

We evaluate CureFuzz by applying it to well-known SDMs that use various learning algorithms. The algorithms include Deep Neural Networks (DNN) [52], Deep Reinforcement Learning (DRL) [3], Multi-agent DRL (MARL) [17], and Imitation Learning (IL) [32]. We also consider a range of sequential decision-making problems. (i.e., autonomous driving [70], aircraft collision avoidance [35], and video game playing [41, 45]). The experimental results demonstrate that CureFuzz effectively and efficiently identifies a significant number of catastrophic failures across all considered SDMs. Overall, CureFuzz outperforms the state-of-the-art methods and detects a more diverse set of crash-triggering scenarios. Furthermore, we have also shown that the crash-triggering scenarios identified by CureFuzz can be utilized to repair the SDMs. By re-running CureFuzz on the repaired SDMs, the number of detected faults decreases by 73%, highlighting the practical utility of our approach in enhancing the effectiveness of SDMs.

The contributions of this paper are summarized as follows:

- We introduce CureFuzz, the first curiosity-driven black-box fuzz testing framework for DL-based sequential decision makers. CureFuzz aims to reveal a diverse set of crash-triggering scenarios, enhancing the safety and effectiveness of these decision-making systems.
- We propose a novel curiosity mechanism that leverages the prediction error of two neural networks to measure the novelty of scenarios for fuzz testing.
- To evaluate the effectiveness of CureFuzz, we conducted experiments on 4 sequential decision-making tasks. The results demonstrate that CureFuzz successfully uncovers crash-triggering scenarios and outperforms our baseline method by a substantial margin.

2 PRELIMINARIES

2.1 Markov Decision Process

The Markov Decision Process (MDP) is a well-known mathematical framework for modeling complex sequential decision-making problems under various uncertainties [61]. In this paper, we focus on the SDMs solving MDPs. An agent (i.e., defined as SDM) and the environment are the two main components of an MDP. In a general paradigm, an agent actively engages with its environment through a sequence of actions. Upon executing an action, the environment responds by transitioning to a new state and provides the agent with feedback in the form of rewards. This reward signal serves as a measure of the quality of the agent's action. The agent's ultimate objective is to learn an optimal policy, which is a strategy that guides its decision-making process to maximize the accumulated reward over time. For simplification, the interactions are assumed to be performed in discrete time steps. Given $t = 0, 1, 2, \ldots$ denotes the discrete time step, we formally define the Markov Decision Process as a tuple of (S, A, T, R), where:

- S is the state space, a set of states that represents all the possible statuses of the environment. A state $s \in S$ refers to the current situation of the agent, and s_t refers to the state at time t.
- A is the action space, a set of available actions that can be taken by the agent. Given a state s, the agent selects its action $a \in A$, accordingly.
- T is the state transition probability function of the environment, such that $T(s'|s, a)$ describes the probability of transitioning to s' from s when the action a is taken.
- R is the reward function. $R(s, a, s')$ refers to the immediate reward received by the agent when it takes the action a at state s and reaches s'.

A sequential decision-making problem is an MDP if and only if it satisfies the Markov Property. In other words, the decision-making process in this environment depends solely on the current state of the environment and not on the sequence of past states. Mathematically, **Markov Property** is described as:

$$P(S_{t+1} = s_{t+1}|S_t = s_t, ..., S_0 = s_0) = P(S_{t+1} = s_{t+1}|S_t = s_t)$$

P denotes the probability of transitioning from $S_t = s_t$ at time step t to $S_{t+1} = s_{t+1}$ at time step $t + 1$. Intuitively, the Markov property ensures that the current state encapsulates all relevant information from history and that the future of the process is independent of the past when the present state is known.

2.2 Sequential Decision Makers

Sequential Decision Makers (SDMs) powered by deep learning have shown strong capabilities in solving MDPs. Here, we introduce the technology used in four state-of-the-art DL-based SDMs: Deep Neural Networks (DNN) [52], Deep Reinforcement Learning (DRL) [3], Multi-agent DRL (MARL) [17], and Imitation Learning (IL) [32].

2.2.1 Deep Neural Network . Although all four techniques involve neural networks, we refer to DNN [52] when the training process is conducted in the supervised setting. In this case, the optimal actions to take for certain states are labeled. During the training session, the model learns to derive the optimal policy by minimizing the difference between its predicted actions and the manually labeled actions. One application is the aircraft collision avoidance system, ACAS Xu [35].

2.2.2 Deep Reinforcement Learning. DRL [51] combines the notion of deep learning with traditional reinforcement learning techniques. Instead of learning with labeled data, a DRL agent learns optimal policy through trial-and-error with the rewards or penalties received from the environment. Algorithms like DQN [51], PPO [63], and TQC [41] have been used in a wide range of tasks. For example, AlphaGo [64], and OpenAI's DOTA 2 agents [81].

2.2.3 Multi-agent Reinforcement Learning. MARL extends conventional reinforcement learning to scenarios that have multiple agents. MARL is a type of machine learning in which multiple agents learn to interact with each other in a shared environment through a feedback mechanism. In MARL, agents learn to make decisions also based on the actions of other agents in the environment. A MARL game can be cooperative, competitive, or a mix of both. In a cooperative game, the agents work together towards a common goal, where the success of one agent is dependent on the success of the other agents.

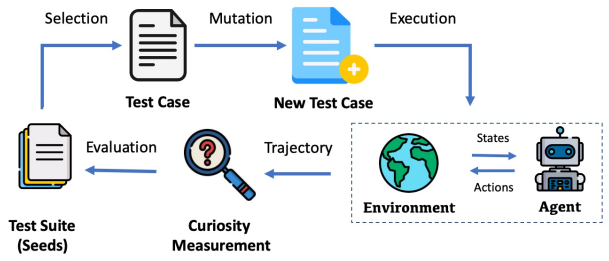


Figure 1. Illustration of the overall workflow of the proposed approach CureFuzz.

2.2.4 Imitation Learning. IL is widely used in scenarios where reinforcement learning may be too slow or expert demonstrations are available, such as autonomous driving. IL usually involves two agents: an expert agent and a student agent. It can be considered as another form of supervised learning, where the student agent aims to mimic the behaviors of the expert agent.

2.3 Fuzz Testing

Fuzz testing is a widely used method in software testing that automates the generation of inputs to identify vulnerabilities, crashes, or any other unexpected behavior in software programs. This approach usually creates a wide range of variants from a set of initial inputs (often referred to as seed corpus). These variants are generated by applying certain mutation operations, such as bit flipping [14]. The mutated inputs are then used to test the software, with the intent of causing unexpected behavior or crashes. The advantage of mutation-based fuzzing lies in its capability to explore the program execution paths that might not be covered under standard testing procedures, thereby increasing the robustness and security of the software.

3 APPROACH

3.1 Assumption

We focus on environments where the transition dynamics satisfy the Markov Property [61], and our testing subjects are DL-based SDMs (agents) solving MDPs. CureFuzz performs fuzz testing in a black-box manner, which is realistic and practical. While white-box testing can be valuable for certain tests, the high complexity of deep learning models, potentially with millions of parameters, can make white-box testing overwhelming. Additionally, in the real world, SDMs' internals are often not available (e.g., users access the SDM through a third-party vendor's service). More specifically, CureFuzz does not require access to the SDM's internal logic, nor knowledge of the environment's transition dynamics or reward mechanism. Moreover, the SDM's policy remains fixed and will not be updated during the fuzzing process. Given a particular state, CureFuzz can obtain the corresponding action from the SDM by interacting with the environment.

CureFuzz is designed to uncover *crash-triggering scenarios* that eventually lead to the crash of SDMs. It is crucial to note that the definition of a 'crash' can vary across different environments. For instance, within the context of autonomous driving, a crash could be defined as an incident where an autonomous vehicle collides with pedestrians. In the context of robotics control, a crash can refer to the falling of a walking robot. We employ the term 'crash' to represent its broader conceptual meaning. Furthermore, our methodology focuses on catastrophic failure rather than minor deviations from optimal performance. If an SDM initiates from a particular scenario, and the subsequent cumulative reward falls below a predefined threshold, yet no catastrophic failure is observed, we do not classify this as a failed case.

3.2 Approach Overview

CureFuzz consists of the following stages: Setup and initialization, seed energy estimation, seed selection, seed mutation, and seed evaluation. Algorithm 2 presents the high-level workflow of CureFuzz and visually represented in Figure 1. As it core, CureFuzz maintains a corpus of seeds, each representing a unique scenario in the environment. We refer to the starting condition of a specific scenario as the initial state. We then observe the resultant actions of the SDM under the initial state. The SDM's actions change the environment and lead to new states, thus we obtain the induced state sequence for each initial state. The interaction between the SDM and the environment is automatically terminated if the state sequence reaches its maximum length or a crash is detected. CureFuzz then estimates the energy of a seed based on its intrinsic reward (to be described later in Section 3.3), the probability of triggering new scenarios (to be described later in Section 3.4), and the probability of triggering a crash (to be described later in Section 3.4). Seeds with higher energy, indicating either their novelty or higher crash probability, are selected preferentially for mutation. This mutation process produces new seeds, whose induced state sequences are then evaluated. This cycle repeats throughout the fuzz testing procedure.

3.3 A Curiosity-driven Search Strategy

Inspired by the success of exploration engineering [42], we propose a curiosity mechanism that measures the novelty of a scenario. Algorithm 1 shows the pseudocode for our curiosity mechanism. Our curiosity module consists of a pair of neural networks, which includes a fixed target network T and a learnable predictor network P. Both networks have identical neural architectures, consisting of a simple multi-layer perceptron (MLP) [23]. The fixed target network is initialized with random weights, which remain unchanged throughout the fuzzing process. In contrast, the predictor network is trained to approximate the output of the target network. For each encountered state, we denote the output generated by the target network as $T(s)$ and the output generated by the predictor network as $P(s)$. The difference between the outputs of these two networks, referred to as the prediction error, serves as a proxy for the novelty of a given state. The prediction error, essentially is the mean squared error (MSE) between the outputs of the target and predictor networks. It is computed as:

$$E(s) = ||T(s) - P(s)||^2 \tag{1}$$

Algorithm 1: The Curiosity Mechanism

```
1  Function initCuriosity():
2      Initialize target network $\phi_{target}$
3      Initialize predictor network $\phi_{pred}$ with same architecture as
         $\phi_{target}$
4      Fix the weights of $\phi_{target}$ to random values
5      return $\phi_{target}, \phi_{target}$
6  Note that $S = \{s_1, s_2, ..., s_t\}$
7  Function Curiosity($S, \phi_{target}, \phi_{target}$):
8      $intrinsic\_reward = 0$
9      for $i = 1$ to $t$ do
           /* Compute intrinsic reward */
10         $r_i = \|\phi_{target}(s'_i) - \phi_{pred}(s'_i)\|^2$
11         $intrinsic\_reward += r_i$
           /* Update predictor network */
12         Update $\phi_{pred}$ to minimize $\|\phi_{target}(s'_i) - \phi_{pred}(s'_i)\|^2$
13     end
14     $intrinsic\_reward = intrinsic\_reward \times \frac{1}{t}$
15     return $intrinsic\_reward, \phi_{target}, \phi_{target}$
```

This prediction error serves as an intrinsic reward signal for each state. In CureFuzz, given the induced states sequence of a scenario, we calculate the intrinsic reward for all states within the sequence. The novelty score of a scenario is the mean of all intrinsic rewards. The intrinsic reward for the 'novel' inputs should be higher than the previously encountered inputs during the fuzzing process, thereby driving the fuzzer to explore those novel states. In addition, when updating the parameters of the predictor network, we apply L_2 regularization to avoid overfitting.

The quantity of training data will influence the magnitude of prediction errors, which explains why our curiosity mechanism serves as a reliable novelty measure for scenarios. When the predictor network encounters few instances for certain types of scenario during training, the prediction error tends to be high. Conversely, when the SDM frequently encounters one type of scenario, the predictor network has more opportunities to learn and mimic the target network's responses for that specific scenario. The predictor network can accurately predict the outcomes and the prediction error tends to be lower, indicating these are familiar scenarios. By leveraging the prediction error as a measure of curiosity, our mechanism can effectively identify scenarios that deviate from the learned patterns, highlighting their novelty. This approach enables the fuzzer to prioritize exploring unfamiliar scenarios, leading to an increased diversity of crash-triggering scenarios during fuzz testing for SDMs.

3.4 CureFuzz Architecture

Setup and Initialization: Lines 13-15 in Algorithm 2 shows the initialization of CureFuzz. The function *EnvMonitor* is responsible for monitoring the interaction between the SDM and the environment. For the target SDM and environment, given an initial state as the input, *EnvMonitor* returns the induced state sequence and the associated cumulative reward. CureFuzz first loads the target SDM and the environment. θ^T and θ^P are the parameters of our curiosity module and are randomly initialized. The fuzzer then generates an

Algorithm 2: CureFuzz Workflow

```
Input   : Target SDM: $SDM$, Environment: $env$
Output  : Crash-triggering Seeds: $C$
1  Function EnvMonitor($env, SDM, s$):
2      $\{s'\}, r, \longleftarrow$ EnvSim($env, SDM, s, max\_step$);
3      return $\{s'\}, r$;
4  Function InitCorpus():
5      $I \longleftarrow \emptyset$
6      while runnign time $<$ 2 hours do
7          $s \longleftarrow RandomState(env)$;
8          $s' \longleftarrow$ EnvMonitor(env, SDM, s);
9          $I \longleftarrow I \cup \{s'\}$;
10     end
11     return $I$;
12 Function CureFuzz($env, SDM$):
13     $C \longleftarrow \emptyset$;
14     $\theta^T, \theta^P \longleftarrow$ initCuriosity;
15     $I \longleftarrow$ InitCorpus(env, SDM);
16     while runnign time $<$ 12 hours do
17         $s \longleftarrow$ SeedSelection($I$);
18         $s_\delta \longleftarrow$ SeedMutation($s$);
19         $s'_\delta \longleftarrow$ EnvMonitor($env, SDM, s_\delta$);
20         $in\_reward, \theta^T, \theta^P \longleftarrow$ Curiosity($s'_\delta, \theta^T, \theta^P$);
21         if isCrashed ($s'_\delta$) then
22             $C \longleftarrow C \cup s'_\delta$;
23         else if isInteresting ($s'_\delta$) then
24             $I \longleftarrow I \cup s'_\delta$;
25     end
26     return $C$;
```

Algorithm 3: Robustness Estimation

```
Input   : State sequence: $S, env, SDM$
Output  : Robustness Measure: $R$
1  Function Robustness($S$):
2      Obtain $s_0$ from $S$
3      $s_0^\delta \longleftarrow$ SeedMutation($s_0$)
4      $S^\delta, r \longleftarrow$ EnvMonitor ($env, SDM, s_0^\delta,$)
5      return $|S_{-1} - S_{-1}^\delta|$
```

initial corpus of seed by randomly sampling with the environment, as depicted in Function *InitCorpus* (Algorithm 2 Line 5 to 11). The underlying assumption is that we are aware of the legitimate state space of the environment, allowing us to generate valid seeds across this state space randomly. Since the definition of legitimate state space varies in different environments, a detailed description for each environment is given in Section 4.1.

Seed energy estimation: The main fuzzing process starts at Line 15. Following the settings from prior works [36, 56], a 12-hour long fuzzing is conducted. At each iteration, a seed is selected from the seed corpus. Similar to traditional fuzz testing of software [67], we first estimate the energy of each seed. A seed with a high energy is more likely to be selected. Because CureFuzz aims to find a diverse set of crashes, the energy of a seed cannot only reflect the novelty but also needs to consider the probability of triggering crashes. To balance these objectives, the estimation process is based on multiple factors. These factors are intrinsic reward (novelty measure), cumulative reward (probability of triggering crashes), and robustness (probability of triggering unseen states).

Cumulative reward. The cumulative reward is a direct measurement on the performance of the SDM. We begin with a simple assumption: if the SDM does not perform well under certain scenarios, mutating on these scenarios is more likely to trigger crashes. CureFuzz prioritizes the seeds with low cumulative reward. Thus a seed with a lower cumulative reward is accordingly given a higher energy estimation.

Robustness. We define the term Robustness as the probability for a given seed to trigger diverse consequences. The calculation of robustness is detailed in the Algorithm 3. Given a seed s and its induced state sequence S, we first record the final state of S, denoted as S_{-1}. We add a tiny random perturbation on s to generate a new initial state s^δ. The mutated state s^δ then is fed the function *EnvMonitor*, and we obtain its induced sequence S^δ. Robustness is then measured as the Euclidean distance between the final states of S and S^δ, which essentially is $|S_{-1} - S_{-1}^\delta|$.

Robustness is a measure of how sensitive an SDM's behavior is to slight perturbations in the original state. When the Euclidean distance between the final states of the original and mutated state sequences is large, it indicates that even a small perturbation in the initial state leads to significantly different outcomes. This suggests that the seed has the potential to trigger a diverse range of behaviors, making it a valuable candidate for further exploration and testing. On the other hand, if the distance between the final states is small, it implies that the SDM's behavior is relatively consistent and less sensitive to small changes in the input. In this case, the seed might be less likely to reveal new or unexpected system behaviors.

Intrinsic reward. The intrinsic reward generated by the curiosity mechanism serves as the novelty measure of a given seed. The intrinsic reward can be easily combined with the cumulative reward and robustness score measurement.

Denoting reward as r, intrinsic reward as i, robustness as r', α, β, and γ as scaling factors, the overall score of seed is:

$$E(s) = e^{-\alpha r} + e^{\beta i} + \gamma r' \tag{2}$$

Seed Selection: We prioritize selecting the seed with a high energy. Given the corpus C and total number of seeds in the corpus N, each seed would be selected with a probability of $\frac{E(s)}{\sum_{i=1}^{N} E(s_i)}$, where $\sum_{i=1}^{N} E(s_i)$ denotes the total energy of the seeds in the corpus.

Seed Mutation: Once a seed is selected from the corpus, the *Seed-Mutation* function generates a new mutated state by applying a small random perturbation to a selected state. The mutated state is fed into the *EnvMonitor* function to generate its corresponding state sequence and collect the cumulative reward. We need to ensure that the mutated seed lies in the legitimate state spaces , and the mutated seed would not trigger an initial crash. In our experiments, we have carefully addressed this concern and verified the validity of each mutated seed, details are given in Section 4.

Seed Evaluation: The function *Curiosity* assigns the intrinsic reward to the newly induced state sequence, and this intrinsic reward serves as the curiosity of the fuzzer in the further mutation of this seed. For each state sequence, we calculate the difference between the two networks of the curiosity mechanism using MSE loss. The parameters of the predictor network are updated using this loss. CureFuzz then checks on the termination status of the state sequence. If a crash is found, the mutated seed is added to the list of crashes, and the fuzzing process continues with the next seed input. When CureFuzz does not identify a crash, it shifts focus to assessing the state sequence based on the induced intrinsic reward and cumulative reward. CureFuzz measures the intrinsic reward of the state sequence. If this reward surpasses a pre-set threshold, the seed responsible for this sequence is considered significant. Consequently, it is added to the seed corpus for further analysis. CureFuzz also compares the cumulative reward of the mutated seed against that of the original seed. This mutated seed is also added to the corpus if the mutated seed's cumulative reward is lower. If neither of these conditions are met, the mutated seed is then discarded, and CureFuzz progresses to the next iteration.

Termination: The fuzzing process continues the above-mentioned iteration and is terminated when the time has exceeded the specified limit (12 hours). Finally, CureFuzz returns the list of crashes found during the fuzzing process.

4 EXPERIMENTAL SETTING

4.1 Research Questions

To evaluate the performance of CureFuzz and comprehensively understand the impact, we formulate the three research questions:
RQ1: How effective is CureFuzz in finding crash-triggering scenarios?
RQ2: Can CureFuzz be effectively guided with the curiosity mechanism?
RQ3: Can we use the crashes found by CureFuzz to improve SDMs?

4.2 Experiment Subject and Environment

We evaluate CureFuzz using the same environments and SDMs as Pang et al. [56]. Our investigation covers various environments, including the CARLA autonomous driving simulator [16], ACAS Xu for collision avoidance in aviation [49], the Cooperative Navigation (Coop Navi) environment for multi-agent reinforcement learning [45], and the BipedalWalker environment in OpenAI Gym [41].

4.2.1 ***Autonomous Driving.*** CARLA [16] is a widely used open-source simulator for autonomous driving research. In the CARLA simulator, at each timestep, the SDM receives an RGB image and its current velocity as inputs. Using this information, the SDM calculates the appropriate steering, throttle, and brake commands to navigate toward the specified goals. The performance of the SDM is assessed in an urban driving environment that includes intersections and traffic lights. Two SDMs are considered in the CARLA environment, which are developed using DRL and IL, respectively. In CARLA, CureFuzz checks for the situations when the SDM-controlled vehicle experiences a collision with other vehicles or buildings. The environment of CARLA can be described as the positions of angles of all vehicles on the map, including the SDM-controlled one. When CureFuzz mutates a given state, small perturbations are randomly generated and added to the positions and angles of these vehicles. We use the CARLA simulator itself to check for the validity of the mutated state. All illegally mutated states and the states that trigger initial collision are discarded in our experiments.

4.2.2 ***Aircraft Collision Avoidance.*** ACAS Xu is a collision avoidance system for the aviation industry [49]. We utilize the popular DNN-based variant of ACAS Xu [35], which has also been broadly studied by previous literature [74]. The system employs 45 distinct neural networks to predict the most appropriate actions to avoid the collision, such as Clear-of-Conflict (which goes straight), weak left (1.5 deg/s), strong left (3.0 deg/s), weak right, and strong right. In ACAS Xu, CureFuzz simply aims to find the scenarios when there are collisions between the SDM-controlled airplane with other airplanes. For mutation, the initial positions and speeds of the SDM-controlled and the other planes slightly changed. The maximal speed of all airplanes is capped at 1,100 ft/sec. Given the predefined range of acceptable states, states that fall outside legal space are automatically discarded.

4.2.3 ***Video Game.*** Coop Navi [45] is an OpenAI environment designed for multi-agent reinforcement learning (MARL) applications. In Coop Navi, agents aim to learn how to cooperate with each other to reach the pre-determined landmarks without colliding with one another. The underlying SDM for this game is proposed by the original publication [45]. This SDM is aware of the position of each agent and the target landmarks, then the SDM decides the moving direction and speed for each agent. CureFuzz aims to find the scenarios when there are collisions between SDM-controlled agents. The initial positions of the three MARL-controlled agents are mutated, and their initial speeds are set to 0 to avoid initial collision. With a clear definition of permissible positions and velocities, CureFuzz ignores any illegal states.

BipedalWalker [41] is an environment in the OpenAI Gym framework that challenges a two-legged robot to navigate through various terrains and obstacles using bipedal locomotion. We select the SDM which is employed with Twin Delayed DDPG with Quantile Distributional Critics (TQC) [41] algorithm, and its implementation is available in the stable-baselines3 repository [71]. The robot takes in a 24-dimensional state and predicts the speed for each leg based on body angle, leg angles, speed, and lidar data. We aim to find the scenarios when the robot falls. Following Pang et al. [56], we mutate the sequence of ground types the robot encounters. Specifically, we make sure that the first 20 frames are "flat" to prevent initial failure. We also place a "flat" between two hurdles to enable the agent to pass the obstacles while taking optimal actions. Since the valid ground types are known, illegal states can be easily detected and discarded.

4.3 Implementation

CureFuzz is coded in Python and the curiosity module is implemented with the Pytorch Library [57]. The curiosity module utilizes a simple multilayer perceptron [23] as the underlying neural architecture for both the target network and predictor network, and we use the ReLU function [1] as the activation function. Following prior work [56], we randomly sample for 2 hours to construct the initial corpus, and the main fuzzing process is conducted for 12 hours as the standard setting [36]. The interaction time between the SDMs and the environment of Coop Navi and Acas Xu takes significantly less time than the other environments. We follow Pang et al. [56] to modify the experiment setup accordingly. Specifically, we reduce the time taken to construct the initial corpus to 1 hour in ACAS Xu, and 30 minutes in Coop Navi, then conduct the 12 hours of fuzzing. For each SDM, we make sure CureFuzz and MDPFuzz are experimented under the same setting to make a fair comparison. When mutating a seed, the magnitude of the random perturbation is a critical factor that can impact the performance of the fuzzer. We re-use the code implementation from Pang et al. for generating mutations. Our experiments are conducted on a server with one Intel Xeon E5-2698 v4 @ 2.20GHz CPU, 504GB RAM, and NVIDIA Tesla V100 GPU.

Table 1. State Coverage of CureFuzz and baselines with 95% confidence interval margins of error.

	Method	5 bins	10 bins	100 bins
Carla (RL)	**MDPFuzz**	7.3% ± 0.6%	1.3% ± 0.1%	$3e^{-5} \pm 3e^{-6}$
	G-Model	5.0% ± 1.2%	1.6% ± 0.3%	$2.4e^{-5} \pm 5.5e^{-6}$
	CureFuzz	8.0% ± 0.9%	3.4% ± 0.3%	$1e^{-4} \pm 2e^{-5}$
ACAS Xu (DNN)	**MDPFuzz**	5.3% ± 0.5%	0.48% ± 0.01%	$2e^{-6} \pm 4e^{-8}$
	G-Model	6.7% ± 0.9%	0.72% ± 0.02%	$3e^{-6} \pm 1e^{-7}$
	CureFuzz	11% ± 0.6%	1.2% ± 0.15%	$6e^{-6} \pm 1e^{-6}$
Carla(IL)	**MDPFuzz**	6.4% ± 1.2%	1.0% ± 0.6%	$2e^{-5} \pm 2e^{-6}$
	G-Model	6.7% ± 0.5%	1.6% ± 0.5%	$3e^{-5} \pm 1e^{-6}$
	CureFuzz	7.2% ± 0.1%	1.6% ± 0.44%	$3e^{-5} \pm 1e^{-7}$
Coop Navi (MARL)	**MDPFuzz**	0.034% ± 0.008%	$3e^{-7} \pm 1e^{-8}$	$5e^{-19} \pm 6e^{-20}$
	G-Model	0.021% ± 0.008%	$6e^{-8} \pm 3e^{-9}$	$7e^{-20} \pm 3e^{-21}$
	CureFuzz	0.059% ± 0.018%	$7e^{-7} \pm 1e^{-7}$	$1e^{-18} \pm 5e^{-19}$
BipedalWalker (DRL)	**MDPFuzz**	0.065% ± 0.002%	–	–
	G-Model	0.011% ± 0.002%	–	–
	CureFuzz	0.42% ± 0.02%	–	–

5 EXPERIMENT RESULTS

RQ1: How effective is CureFuzz in finding crash-triggering scenarios?

We compare the performance of CureFuzz with two state-of-the-art approaches: MDPFuzz [56] and the method proposed by Li et al. [86] (referred to as "G-Model" in our paper). MDPFuzz is a black-box fuzz testing framework for SDMs while G-Model employs a model-based method to generate diverse scenarios. Our evaluation focuses on three key metrics: environmental state coverage, the total number of detected crash-triggering scenarios, and the distinct types of crash-triggering scenarios. We repeat the experiment five times for each SDM and report the average results. The inclusion of two markedly different baselines and three diverse metrics ensures that our evaluation comprehensively compares the effectiveness of our CureFuzz and the baselines from multiple perspectives.

Coverage Analysis. Coverage analysis plays a crucial role in evaluating the thoroughness and completeness of a testing technique. Inspired by fuzzing for traditional software, which measures code coverage, we measure the environmental state coverage in our experiments. State coverage refers to the effectiveness of a fuzzer in covering the possible scenarios within the state space of the environment. Considering that experimental environments have a high-dimensional and continuous state space, to calculate state coverage, we discretize the state space of the experimental environments by bucketing continuous variables into discrete segments. In other words, we divide each dimension of the state space into a fixed number of bins, effectively creating a grid over the state space. Each bin represents a discrete state. When performing the state discretization, we use multiple numbers of bins (i.e., 5, 10, 100) [44]. Note that the exception here is the case of BipedalWalker, where we mutate the sequence of ground types that the robot needs to walk through. Since the available number of ground types is fixed, we only need to calculate the proportion of encountered ground types over the total number of ground types as state coverage and there are no actual bins. We present the average state coverage results of CureFuzz and the baselines in Table 1. CureFuzz demonstrates promising state coverage performance in comparison to baseline methods across various environments and bin numbers. We perform the Mann-Whitney U [54] statistical significance test at 95% significance level and compute effect size measure (Cohen's d [22]). Notable environments where it significantly and substantially outperforms others include Carla(RL), Acas Xu (DNN), Coop Navi(MARL), and Bipedal-Walker(RL). The only exception is in Carla(IL), where CureFuzz shows comparable results to G-Model. Overall, CureFuzz achieves the best coverage rate.

Total Number of Crashes. Table 2 shows the comparative results of CureFuzz and baseline methods in terms of the number of detected crashes. The results show that CureFuzz consistently outperforms MDPFuzz in all five sets of experiments. The most significant improvement is observed in the SDM of BipedalWalker(RL), with a remarkable 422.22% increase in detected crashes, rising from 126 to 658. For CARLA(RL) and CARLA(IL), the improvements are 145.83% (from 120 to 295) and 110.47% (from 86 to 181), respectively; for Coop Navi(MARL), the performance is boosted by 62.21% (from 52.4 to 85). The smallest improvement is for ACAS Xu(DNN), which is 46.45% (from 183 to 268). Furthermore, CureFuzz outperforms G-Model for four out of five SDMs by up to 134.16% (from 281 to 658 in BipedalWalker(RL)). For Carla(RL) and Carla(IL), the improvements are 86.7% (from 158 to 195)and 60.1% (from 113 to 181). CureFuzz also archives an improvement of 94.2% in Acas Xu(DNN). After performing the statistical test and computing effect size, results show that CureFuzz statistically significantly and substantially outperforms both baselines in these cases. The only exception is in the context of Coop Navi(MARL), where G-Model identifies more crashes than CureFuzz (185.4 vs. 85). We perform additional analysis to investigate why it is happening for Coop Navi(MARL). Recall that the goal of the Coop Navi game is to find the scenarios when the agents collide with each other, in Figure 2, we select the median performance from the five repeated experiments for CureFuzz and G-Model, and plot the positions of the agents in the crash-triggering scenarios found by different methods. While the valid range of the agent's position is from -1 to 1, from Figure 2, we can see that G-Model tends to find the crashes that the agents are positioned around the boundary of the valid input. In contrast, CureFuzz covers more spread positions. It also explains why the coverage of G-Model is lower than CureFuzz but it finds more crashes for Coop Navi (MARL). CureFuzz and G-Model tend to be interested in a different area of the search space, thus one possible future direction could combine the advantages of both kinds of methods.

Distinct Crashes. Using the same state discretization procedure that we used for coverage analysis, we transform the continuous state space into a discrete grid representation again. For each method,

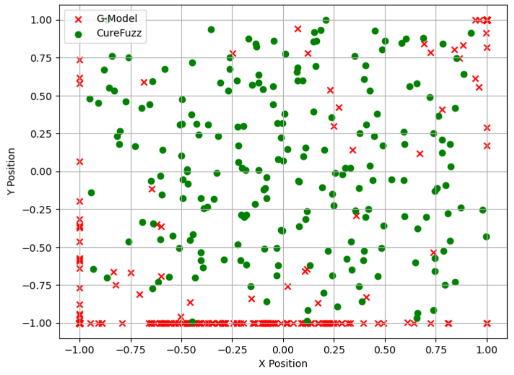


Figure 2. Visualization of Agent Positions in Crash-Triggering Scenarios found by various methods for Coop Navi (MARL). This graph plots the x and y coordinates, ranging from -1 to 1. The red crosses represent G-model, the green dots represent CureFuzz, and the blue triangles represent MDPFuzz.

Table 2. Average total number of crashes found by CureFuzz and the baselines run across different SDMs and margin of errors in 95% confidence interval.

	Method	Mean
Carla (RL)	**MDPFuzz**	120 ± 22.8
	G-Model	158 ± 18.6
	CureFuzz	295 ± 38.3
ACAS Xu (DNN)	**MDPFuzz**	183 ± 15.9
	G-Model	138 ± 27.9
	CureFuzz	268 ± 26.8
Carla (IL)	**MDPFuzz**	86 ± 25.0
	G-Model	113 ± 33.5
	CureFuzz	181 ± 25.4
Coop Navi (MARL)	**MDPFuzz**	52.4 ± 8.8
	G-Model	185.4 ± 36.6
	CureFuzz	85 ± 7.3
BipedalWalker (RL)	**MDPFuzz**	126 ± 31.8
	G-Model	281 ± 52.5
	CureFuzz	658 ± 98.3

we determine the average number of unique grid cells occupied by the detected crashes. We account for one grid cell as one distinct crash. As shown in Table 3, CureFuzz finds more types of crash-triggering scenarios than MDPFuzz in all bin numbers and every SDM. For the 100 bins situation, the largest improvement is seen for the Carla(IL) SDM, where CureFuzz found 200% more types than MDPFuzz. Similarly, for the other SDMs, CureFuzz found 88.9%, 177.8%, and 62.8% more types of crashes respectively. After performing the statistical test and computing effect size, we find that CureFuzz statistically significantly and substantially outperforms the baselines apart from Coop Navi (MARL). Again, in the context of Coop Navi (MARL), CureFuzz falls short of G-Model. We have discussed this in detail in the sub-section above.

Efficiency Analysis. We further conduct an in-depth analysis to investigate the efficiency of CureFuzz. The total time taken by a fuzz testing method can be split into two parts, which are execution time and analysis time. The execution time corresponds to the

Table 3. Average distinct types of crash-triggering scenarios found by CureFuzz and baselines run across different SDMs and margin of errors in 95% confidence interval.

	Method	5 bins	10 bins	100 bins
Carla (RL)	**MDPFuzz**	9.0 ± 0.9	17.8 ± 2.4	29.8 ± 3.9
	G-Model	6.8 ± 1.0	12.2 ± 1.1	20.6 ± 1.5
	CureFuzz	10.0 ± 0.6	30.2 ± 3.1	89.4 ± 25.3
ACAS Xu (DNN)	**MDPFuzz**	8.0 ± 0.9	9.0 ± 0.9	9.0 ± 0.9
	G-Model	5.8 ± 1.6	6.4 ± 1.1	6.6 ± 1.7
	CureFuzz	12.0 ± 0.9	13.6 ± 0.7	17.0 ± 1.5
Carla (IL)	**MDPFuzz**	9.0 ± 0.9	14.4 ± 2.3	21.6 ± 3.0
	G-Model	5.6 ± 0.7	12.8 ± 3.2	23.4 ± 6.0
	CureFuzz	10.0 ± 0.0	27.5 ± 1.9	60.0 ± 10.5
Coop Navi (MARL)	**MDPFuzz**	52 ± 11.0	52.4 ± 11.7	52.4 ± 11.7
	G-Model	104.8 ± 35.5	156.2 ± 30.0	184.6 ± 38.8
	CureFuzz	83.5 ± 8.0	85.3 ± 7.3	85.3 ± 7.3
BipedalWalker (RL)	**MDPFuzz**	126 ± 31.8	-	-
	G-Model	281 ± 52.5	-	-
	CureFuzz	658 ± 98.3	-	-

Table 4. Time comparison between CureFuzz and MDPFuzz in seconds.

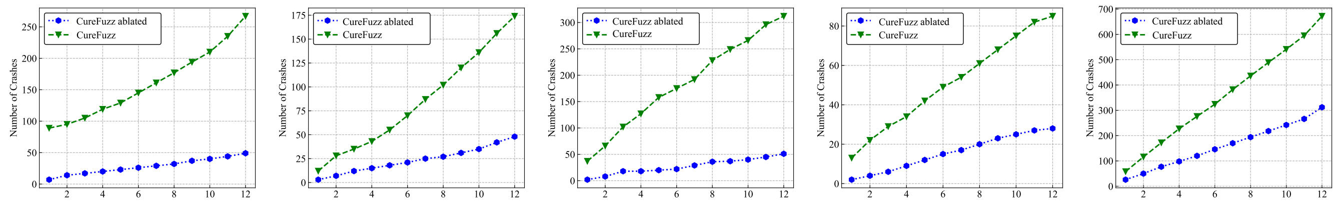
	MDPFuzz	**CureFuzz**
Carla (RL)	0.855	0.008 **(+99.1%)**
ACAS Xu (DNN)	0.231	0.005 **(+97.8%)**
Carla (IL)	0.985	0.005 **(+99.5%)**
Coop Navi (MARL)	0.033	0.006 **(+81.8%)**
BipedalWalker (RL)	0.970	0.011 **(+98.9%)**

duration an SDM needs to interact with its environment, while the analysis time accounts for the computational time needed for the fuzzing process (i.e., seed selection and seed evaluation). Table 4 presents the average analysis time for each iteration of CureFuzz and MDPFuzz. Note here we do not compare with G-Model in the efficiency analysis since it operates differently. It is not a fuzzing test method and requires the retraining of its generative model in each iteration. This retraining process can significantly extend the time taken, making it incomparable to the efficiency of CureFuzz and MDPFuzz, as these two fuzzing methods do not necessitate repeated training. The results clearly demonstrate the remarkable efficiency of CureFuzz, with its analysis time typically requiring only up to 0.08 seconds. In contrast, MDPFuzz takes about one second in some cases. On average, CureFuzz's analysis time is 81.8% faster than that of MDPFuzz, highlighting the minimal time overhead associated with our curiosity mechanism across various environments and state spaces.

Answer to RQ1: CureFuzz can efficiently find crash-triggering scenarios across various SDMs and environments.

RQ2: Can CureFuzz be effectively guided with the curiosity mechanism?

For this research question, we follow the same experimental setting with RQ1. We remove the curiosity component from CureFuzz and conduct the fuzzing again on our target SDMs. Table 5 shows the



(a) ACAS Xu(DNN) **(b) Carla(IL)** **(c) Carla(RL)** **(d) Coop Navi(MARL)** **(e) BipedalWalker(RL)**

Figure 3. Results for RQ2: Comparison between CureFuzz with and without the curiosity mechanism. The x-axis represents the time passed in hours and the y-axis represents the number of found crashes. The complete CureFuzz is represented with green color and triangles, the ablated CureFuzz is represented with blue color and circles.

Table 5. Results for RQ2, shows the average state coverage (for 100 bins), total number of crashes, distinct types of crash-triggering scenarios (for 100 bins) found by the ablated CureFuzz

	State Coverage	Total crash	Distinct crash
Carla (RL)	4e-3%	46.25	26.6
Acas Xu (DNN)	2.5e-4%	46.0	7.8
Carla (IL)	1.3e-3%.	52.6	23.8
Coop Navi (MARL)	1.0e-17%	28.8	28.8
BipedalWalker (RL)	2.4e-1%	367	367

performance of the ablated CureFuzz . We can observe that the addition of the curiosity mechanism plays a vital role in boosting the effectiveness of CureFuzz, as it provides an additional signal that motivates CureFuzz to further explore the states it is curious about. We plot the number of crashes found by CureFuzz and the ablated CureFuzz on an hourly basis in Figure 3. Notice that the experiments are repeated five times, and we select the median performer for plotting. We can see that CureFuzz consistently outperforms ablated CureFuzz across all scenarios and time intervals.

Answer to RQ2: The curiosity mechanism can effectively guide CureFuzz. The novelty measure provided by the curiosity mechanism serves as important guidance for CureFuzz to find crash-triggering scenarios.

RQ3: Can we use the crash-triggering scenarios found by CureFuzz to improve SDMs?

In this research question, we investigate whether the identified crash-triggering scenarios found by CureFuzz can advance the overall performance of SDMs. We follow the same procedure conducted by Pang et al. [56] to repair the DNN-based SDM for ACAS Xu with further fine-tuning. In RQ1, the experiments for ACAS Xu are repeated five times for CureFuzz. We select the median performer (ranked by the number of detected crashes), and then manually inspect these crash-triggering scenarios to identify the optimal actions to avoid collisions. In the end, to verify the effectiveness of the repair, we utilize CureFuzz again to test the newly fine-tuned SDM.

For the repaired ACAS Xu, 55 faults and 4 distinct types of crash-triggering scenarios are detected. We find that the crashes found by CureFuzz are reduced by 73% compared with the original DNN model. Therefore, CureFuzz's findings can boost the performance of the SDM. It is important to note that a similar effectiveness enhancement process can also be applied to other SDMs. We demonstrate it in the environment of ACAS Xu, as aviation avoidance is one of the safety-critical situations and the DNN SDM of ACAS Xu has a relatively straightforward architecture. In comparison, previous papers have shown that substantial computational resources are needed to train complex SDMs. For example, the training process of an RL agent under the CARLA environment can take more than a month [15].

Answer to RQ3: The crash-triggering scenarios found by CureFuzz are beneficial for enhancing the effectiveness of SDMs in avoiding catastrophic failures. The experiment result shows that the number of found crashes is reduced by 73% on the repaired model.

6 DISCUSSION

Facilitating Fault Detection through Curiosity Mechanism Optimization. The curiosity mechanism's optimization process can be conceptualized as knowledge distillation from a randomly initialized neural network to the predictor network [11]. The prediction error of the target network and the predictor network then serves as a proxy for measuring uncertainty, specifically epistemic uncertainty [28]. Epistemic uncertainty would be particularly high in regions of the input space where few similar examples were seen in the predictor network's training data. By focusing on areas of high epistemic uncertainty, where the predictor network struggles, the testing process inherently explores a wider variety of scenarios. Previous studies have shown that generating a diverse set of test cases can effectively detect failures [2, 26, 30]. Increasing the diversity of test cases leads to a more comprehensive exploration of the fault space, thereby improving the chances of identifying faults and crash-triggering scenarios [2, 26, 30]. As we showed in RQ2, the guidance provided by the curiosity mechanism effectively enhances CureFuzz's ability to detect a diverse set of crashes. Additionally, a varied set of crash-triggering scenarios is particularly beneficial during the model development phase, as it provides developers with a wider range of contexts for debugging, repairing, and enhancing the model's effectiveness.

Comparison with SDM Testing Methods. While MDPFuzz also utilizes a density-based method to estimate the novelty of the state sequence, it primarily focuses on mutating seeds with high crash potential. In contrast, CureFuzz emphasizes exploring novel scenarios. As a result, we can observe that prioritizing novel scenarios is more effective and can improve the diversity of crash-triggering scenarios. Moreover, our curiosity mechanism is computationally more efficient than the density-based method of MDPFuzz, making it better suited for complex environments. G-Model employs a topological similarity measure that evaluates the novelty of the state to guide the generation of diverse scenarios. However, their proposed novelty measurement is only applied to the termination state instead of the entire state sequence. In comparison, CureFuzz captures richer information from the entire state sequence, which gives a more comprehensive picture of the SDM's behavior over time. This comprehensive approach is particularly crucial in our experiment setting, where SDMs engage in hundreds of interactions with the environment per run. By focusing solely on the termination state, G-Model risks overlooking a wealth of valuable information embedded in the intermediate states.

Comparison with Exploration Heuristics in RL. Exploration is a long-studied topic in the field of reinforcement learning. Apart from leveraging the prediction error, another popular method is the count-based method [6, 68], where the visited frequency of states is leveraged as the novelty measure. However, it may not scale well to high-dimensional state spaces or non-discrete environments where the state space becomes effectively infinite, making state visitation counts sparse and less informative. Our curiosity mechanism overcomes this by using neural network-generated predictions as a basis for exploration, providing a more generalizable measure of novelty. In highly complex environments where the SDM's behavior is influenced by an extensive range of variables and intricate interactions, CureFuzz may struggle to accurately identify all potential failure scenarios. The complexity can mask underlying issues, making them harder to detect.

Real-World Complexities. Another critical part to consider is the discrepancy between our simulated environment and real-world scenarios. Simulation environments, while sophisticated, may not perfectly replicate the complexities of real-world scenarios. For example, environmental and sensor-related noise can influence the accuracy of simulation-based testing compared to real-world scenarios. As pointed out by Stocco et al. [66], this gap between the virtual and physical-world testing may lead to any testers producing false positives and false negatives. Moreover, when applied to larger-scale and more complex systems, the resource requirements of CureFuzz, which mainly come from three aspects, need to be considered. Firstly, the sophistication of the simulator increases, necessitating greater computational resources. Secondly, CureFuzz requires more resources to handle intricate scenarios characterized by numerous properties, such as high-resolution images and radar data. Thirdly, the exploration of a larger input space in extensive systems demands more time and resources. Future research will delve into the effects of applying CureFuzz in these advanced, complex environments

7 THREATS TO VALIDITY

Threats to Internal Validity. These threats relate to factors within the experimental design. To ensure the implementations of baselines are correct, we reuse the official replication package released by Pang et al. [56] and Li et al [86]. To reduce the variability due to the inherent randomness in our experiments, we repeat each experiment five times. We report the mean results marginal of errors and conduct statistical tests.

Threats to External Validity. These threats relate to the generalizability of our experimental results. We mitigated this threat by including typical and various environments, including autonomous driving, aviation collision avoidance systems, and video game playing. We also experiment on five state-of-the-art SDMs, which are powered by different types of complex technologies (i.e., DNN, DRL, MARL, and IL). In the future, we aim to apply our testing approach to more complex environments to gain more profound insights.

Threats to Construct Validity. Threats to construct validity are related to the suitability of our evaluation metrics. The number of faults found per 12 hours is a widely adopted metric for fuzzing [56, 77]. Moreover, we report the state coverage and distinct types of crash-triggering scenarios. Because most of the environments considered in our paper have a continuous state space, we use state discretization to transform a continuous space into a discrete one. The number of bins has a major impact on the state discretization process and is not straightforward to decide. To mitigate this risk, we report the results for multiple bin numbers (i.e., 5, 10, 100). Thus, we believe the associated threats to construct validity are minimal.

8 RELATED WORK

8.1 Fuzz Testing

Fuzz testing is a predominant method to detect vulnerabilities and faults in software engineering [10, 12, 30, 58]. Advancements in fuzzing methodologies have incorporated the usage of execution states. These approaches aim to guide the fuzzing process more effectively by considering the stateful nature of certain applications [30, 34, 55, 60]. For example, due to the complexity of network protocols, network services (i.e., implementations of network protocols) respond differently to identical input messages based on their current session state, leading to stateful bugs that are only activated under specific, often complex, conditions. To address these challenges, the development of stateful black-box [34] and gray-box methods [30, 55, 60] has gained momentum. More recently, inspired by the success of large language models (LLM) in a wide range of SE-related tasks [24, 25, 85], Meng et al. [50] combined the pre-trained LLM to extract information about the protocol that can be used during the fuzzing process. However, these methods are tailored to the discrete state space of network services, while the state spaces encountered in SDMs typically are continuous and high dimensional (i.e., with a large number of features).

8.2 Diversity in Testing

The diversity of input and output is a long-studied problem in software testing. Executing similar test cases results in the execution of identical parts of the source code, which leads to revealing the same faults [7, 13, 27]. Thus, researchers are motivated to propose various testing methods to support diversity. Böhme et al. formulate the fuzzing as a species discovery problem [8–10], where inputs are classified into different species and more energy is assigned to the rare species (e.g., rare paths) to discover new behaviors of the program. Entropic [10] used Shannon's entropy to measure the general rate at which the fuzzer discovers new behaviors. Our objectives in this paper are similar to these studies but in the context of SDM testing. As several studies have shown the effectiveness of diversity metrics in guiding the testing of software systems, we investigated its usefulness in testing SDMs. In recent years, researchers also proposed diversity-based testing methods for Deep Learning models. Aghababaeyan et al. [2] studied the impact of black-box input diversity metrics for testing DNNs. For the image dataset, they found that geometric diversity outperforms white-box coverage criteria in terms of fault detection and computational time. Such methods are designed for images and are not compatible with SDMs. Zohdinasab et al. [87] developed DeepHyperion, a search-based test method that automatically generates a large, diverse set of testing scenarios using illumination search [53]. We did not include DeepHyperion to compare with CureFuzz as DeepHyperion necessitates human expertise to select interpretable features within a given environment. As the effectiveness of DeepHyperion depends on how good these domain-specific features and metrics are designed, and we neither have ready domain-specific features and metrics for our complex environments nor sufficient domain expertise to design them, we did not include DeepHyperion in comparison.

8.3 Deep Learning Testing

In the last decade, Deep Learning-based models have achieved great success in a wide range of tasks [43]. However, the inherent safety concerns of these models limit their application in real life [43]. Consequently, the field of testing Deep Learning-based models has drawn great attention recently. Researchers have proposed various solutions, such us differential testing [4, 79], metamorphic testing [5], and coverage-driven methods [48, 76] for testing various deep learning-based models (e.g., code models [80], autonomous driving [82], etc). Inspired by the use of code coverage in testing traditional software programs, Pei et al. [59] introduced the concept of neuron coverage and proposed DeepXplore to detect behavioral inconsistencies in DNNs. Subsequent research introduced more structured neuron coverage metrics and developed a range of coverage-guided fuzzing tools, such as DeepHunter [76], DeepTest [72], DeepGauge [48], and DeepCT [47] for testing DNNs. However, recent studies suggested that the existing neuron coverage metrics may not be effective in the generation of test cases SDMs [73, 78, 83]. Trujillo et al. [73] studied the relationship between neuron coverage [59] and the performance of DRL agents. More specifically, they focus on investigating the relationship between neuron coverage and rewards of two different models of Deep Q-Network (DQN) [64] in the game of Mountain Car [37]. They demonstrated that a high neuron coverage cannot be related to the achievement of high rewards for DRL agents. Moreover, they also discovered that excessive exploration by the agent can also lead to achieving maximum coverage, which can result in exploring irrelevant actions that do not help the agent maximize its reward.

8.4 SDM Testing

Recently, numerous methods have been proposed to test deep learning-based sequential decision-makers. Lu et al. presented a mutation testing framework for DRL systems [46] and proposed mutation operators that adapt to the DRL systems. Gong et al. [**?**] combine curiosity information and estimate the information of the system under test to produce adversarial policies. STARLA [88] utilized a genetic algorithm to narrow the search space of test cases and it is applied on Deep-Q-Learning agents in the Cartpole and Mountain car environments. We did not include STARLA in our comparison since STARLA is only applicable to Deep-Q-Learning agents and requires the internal logic of the RL model. While CureFuzz can be applied to any kind of SDMs and in a black-box setting. Tapple et al. [69] presented a search-based testing approach for RL agents with stochastic policies. This method uses a depth-first backtracking search algorithm to identify a reference trace that solves the RL task and identifies a set of boundary states that can lead to unsafe states. However, their method is not directly applicable to deterministic policies interacting with stochastic environments, which is common in safety-critical domains. In contrast, CureFuzz targets a broader range of SDMs solving MDPs. Another line of work in testing SDMs utilizes metamorphic testing. Steinmetz et al. [65] and Eniser et al [19]. have pioneered this direction. Steinmetz et al. identify bugs based on the availability of a better-performing alternative policy, while Eniser et al. define a bug as a failure of a Deep Reinforcement Learning (DRL) agent in an easier state, despite success in more complex ones. The cornerstone of these methods is their use of manually designed metamorphic oracles and relaxations, which have proven effective in uncovering bugs. Eisenhut et al. [18] advanced this concept by introducing fully automated test oracles for metamorphic testing. They aim to find a policy that can perform better than the one under test, and call it a bug if such a case is identified. Researchers have also investigated other security issues of SMD, e.g., data poisoning [21], etc.

9 CONCLUSION AND FUTURE WORK

This paper introduces CureFuzz, a curiosity-driven fuzz testing method for SDMs. CureFuzz proposes a curiosity mechanism to measure the novelty of a scenario, which aims to reveal a diverse set of crash-triggering scenarios. CureFuzz demonstrates its effectiveness in various applications such as video game playing, autonomous driving, and aircraft collision avoidance. CureFuzz outperforms the existing state-of-the-art method by a considerable margin in revealing crashes. In the future, we will evaluate the performance of CureFuzz on more environments. A replication package is provided at **https://figshare.com/s/6d7a984f6ef797904d4b**.

ACKNOWLEDGMENTS

This research/project is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-017).

REFERENCES

[1] Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).
[2] Zohreh Aghababaeyan, Manel Abdellatif, Lionel C. Briand, Ramesh S, and Mojtaba Bagherzadeh. 2023. Black-Box Testing of Deep Neural Networks through Test

Case Diversity. *IEEE Trans. Software Eng.* 49, 5 (2023), 3182–3204. https://doi.org/10.1109/TSE.2023.3243522

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.

[4] Muhammad Hilmi Asyrofi, Zhou Yang, and David Lo. 2021. CrossASR++: A Modular Differential Testing Framework for Automatic Speech Recognition. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1575–1579. https://doi.org/10.1145/3468264.3473124

[5] Muhammad Hilmi Asyrofi, Zhou Yang, Imam Nur Bani Yusuf, Hong Jin Kang, Ferdian Thung, and David Lo. 2021. Biasfinder: Metamorphic test generation to uncover bias for sentiment analysis systems. *IEEE Transactions on Software Engineering* 48, 12 (2021), 5087–5101.

[6] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems* 29 (2016).

[7] Robert Binder. 2000. *Testing object-oriented systems: models, patterns, and tools.* Addison-Wesley Professional.

[8] Marcel Böhme. 2018. STADS: Software testing as species discovery. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 27, 2 (2018), 1–52.

[9] Marcel Böhme, Danushka Liyanage, and Valentin Wüstholz. 2021. Estimating residual risk in greybox fuzzing. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 230–241.

[10] Marcel Böhme, Valentin J. M. Manès, and Sang Kil Cha. 2020. Boosting fuzzer efficiency: an information theoretic perspective. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 678–689. https://doi.org/10.1145/3368089.3409748

[11] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. 2019. Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=H1lJJnR5Ym

[12] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2019. Coverage-Based Greybox Fuzzing as Markov Chain. *IEEE Transactions on Software Engineering* 45, 5 (2019), 489–506. https://doi.org/10.1109/TSE.2017.2785841

[13] Emanuela G Cartaxo, Patrícia DL Machado, and Francisco G Oliveira Neto. 2011. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability* 21, 2 (2011), 75–100.

[14] Sang Kil Cha, Maverick Woo, and David Brumley. 2015. Program-adaptive mutational fuzzing. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 725–741.

[15] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. 2020. Learning by Cheating. In *Proceedings of the Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 100)*, Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (Eds.). PMLR, 66–75. https://proceedings.mlr.press/v100/chen20a.html

[16] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings (Proceedings of Machine Learning Research, Vol. 78)*. PMLR, 1–16. http://proceedings.mlr.press/v78/dosovitskiy17a.html

[17] Maxim Egorov. 2016. Multi-agent deep reinforcement learning. *CS231n: convolutional neural networks for visual recognition* (2016), 1–8.

[18] Jan Eisenhut, Alvaro Torralba, Maria Christakis, and Jörg Hoffmann. 2023. Automatic Metamorphic Test Oracles for Action-Policy Testing. (2023).

[19] Hasan Ferit Eniser, Timo P Gros, Valentin Wüstholz, Jörg Hoffmann, and Maria Christakis. 2022. 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*. 52–63.

[20] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. 2021. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters* 6, 3 (2021), 4257–4264.

[21] Chen Gong, Zhou Yang, Yunpeng Bai, Junda He, Jieke Shi, Arunesh Sinha, Bowen Xu, Xinwen Hou, Guoliang Fan, and David Lo. 2022. Mind Your Data! Hiding Backdoors in Offline Reinforcement Learning Datasets. https://doi.org/10.48550/ARXIV.2210.04688

[22] Robert J Grissom and John J Kim. 2005. *Effect sizes for research: A broad practical approach.* Lawrence Erlbaum Associates Publishers.

[23] Simon Haykin. 1994. *Neural networks: a comprehensive foundation.* Prentice Hall PTR.

[24] Junda He, Bowen Xu, Zhou Yang, DongGyun Han, Chengran Yang, and David Lo. 2022. PTM4Tag: Sharpening Tag Recommendation of Stack Overflow Posts with Pre-Trained Models. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension* (Virtual Event) *(ICPC '22)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3524610.3527897

[25] Junda He, Xin Zhou, Bowen Xu, Ting Zhang, Kisub Kim, Zhou Yang, Ferdian Thung, Ivana Clairine Irsan, and David Lo. 2023. Representation Learning for Stack Overflow Posts: How Far Are We? *ACM Trans. Softw. Eng. Methodol.* (dec 2023). https://doi.org/10.1145/3635711 Just Accepted.

[26] Hadi Hemmati, Andrea Arcuri, and Lionel C. Briand. 2013. Achieving scalable model-based testing through test case diversity. *ACM Trans. Softw. Eng. Methodol.* 22, 1 (2013), 6:1–6:42. https://doi.org/10.1145/2430536.2430540

[27] Hadi Hemmati, Zhihan Fang, and Mika V Mantyla. 2015. Prioritizing manual test cases in traditional and rapid release environments. In *2015 IEEE 8th international conference on software testing, verification and validation (ICST)*. IEEE, 1–10.

[28] Eduard Hofer, Martina Kloos, Bernard Krzykacz-Hausmann, Jörg Peschke, and Martin Woltereck. 2002. An approximate epistemic uncertainty analysis approach in the presence of epistemic and aleatory uncertainties. *Reliability Engineering & System Safety* 77, 3 (2002), 229–238.

[29] Ionel-Alexandru Hosu and Traian Rebedea. 2016. Playing Atari Games with Deep Reinforcement Learning and Human Checkpoint Replay. *CoRR* abs/1607.05077 (2016). arXiv:1607.05077 http://arxiv.org/abs/1607.05077

[30] Fan Hu, Shisong Qin, Zheyu Ma, Bodong Zhao, Tingting Yin, and Chao Zhang. 2023. NSFuzz: Towards Efficient and State-Aware Network Service Fuzzing - RCR Report. *ACM Trans. Softw. Eng. Methodol.* 32, 6 (2023), 161:1–161:8. https://doi.org/10.1145/3580599

[31] Yuqi Huai, Sumaya Almanee, Yuntianyi Chen, Xiafa Wu, Qi Alfred Chen, and Joshua Garcia. 2023. scenoRITA: Generating Diverse, Fully Mutable, Test Scenarios for Autonomous Vehicle Planning. *IEEE Transactions on Software Engineering* 49, 10 (2023), 4656–4676. https://doi.org/10.1109/TSE.2023.3309610

[32] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 1–35.

[33] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. 2018. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2034–2039.

[34] William Johansson, Martin Svensson, Ulf E. Larson, Magnus Almgren, and Vincenzo Gulisano. 2014. T-Fuzz: Model-Based Fuzzing for Robustness Testing of Telecommunication Protocols. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. 323–332. https://doi.org/10.1109/ICST.2014.45

[35] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2018. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *CoRR* abs/1810.04240 (2018). arXiv:1810.04240 http://arxiv.org/abs/1810.04240

[36] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. 2018. Evaluating Fuzz Testing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 2123–2138. https://doi.org/10.1145/3243734.3243804

[37] W Bradley Knox, Adam Bradley Setapen, and Peter Stone. 2011. Reinforcement Learning with Human Feedback in Mountain Car.. In *AAAI Spring Symposium: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration*.

[38] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[39] Mario Köppen. 2000. The curse of dimensionality. In *5th online world conference on soft computing in industrial applications (WSC5)*, Vol. 1. 4–8.

[40] Nishanth Kumar. 2020. The Past and Present of Imitation Learning: A Citation Chain Study. *CoRR* abs/2001.02328 (2020). arXiv:2001.02328 http://arxiv.org/abs/2001.02328

[41] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. 2020. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*. PMLR, 5556–5566.

[42] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[43] David Lo. 2023. Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps. *arXiv preprint arXiv:2309.04142* (2023).

[44] Karol Lina López, Christian Gagné, and Marc-André Gardner. 2018. Demand-side management using deep learning for smart charging of electric vehicles. *IEEE Transactions on Smart Grid* 10, 3 (2018), 2683–2691.

[45] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 6379–6390. https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html

[46] Yuteng Lu, Weidi Sun, and Meng Sun. 2022. Towards mutation testing of reinforcement learning systems. *Journal of Systems Architecture* 131 (2022), 102701.

[47] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 614–618.

[48] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 120–131.

[49] Mike Marston and Gabe Baca. 2015. *ACAS-Xu initial self-separation flight tests*. Technical Report.

[50] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large language model guided protocol fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nat.* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

[52] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2018. Methods for interpreting and understanding deep neural networks. *Digital signal processing* 73 (2018), 1–15.

[53] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[54] Nadim Nachar et al. 2008. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology* 4, 1 (2008), 13–20.

[55] Roberto Natella. 2022. StateAFL: Greybox fuzzing for stateful network servers. *Empir. Softw. Eng.* 27, 7 (2022), 191. https://doi.org/10.1007/S10664-022-10233-3

[56] Qi Pang, Yuanyuan Yuan, and Shuai Wang. 2022. MDPFuzz: testing models solving Markov decision processes. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, Sukyoung Ryu and Yannis Smaragdakis (Eds.). ACM, 378–390. https://doi.org/10.1145/3533767.3534388

[57] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[58] Ketan Patil and Aditya Kanade. 2018. Greybox fuzzing as a contextual bandits problem. *arXiv preprint arXiv:1806.03806* (2018).

[59] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[60] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. 2020. AFLNET: A Greybox Fuzzer for Network Protocols. In *13th IEEE International Conference on Software Testing, Validation and Verification, ICST 2020, Porto, Portugal, October 24-28, 2020*. IEEE, 460–465. https://doi.org/10.1109/ICST46399.2020.00062

[61] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science* 2 (1990), 331–434.

[62] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[63] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[64] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.* 529, 7587 (2016), 484–489. https://doi.org/10.1038/nature16961

[65] Marcel Steinmetz, Daniel Fišer, Hasan Ferit Eniser, Patrick Ferber, Timo P Gros, Philippe Heim, Daniel Höller, Xandra Schuler, Valentin Wüstholz, Maria Christakis, et al. 2022. Debugging a Policy: Automatic Action-Policy Testing in AI Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 32. 353–361.

[66] Andrea Stocco, Brian Pulfer, and Paolo Tonella. 2022. Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems. *IEEE Transactions on Software Engineering* (2022).

[67] Ari Takanen, Jared D Demott, Charles Miller, and Atte Kettunen. 2018. *Fuzzing for software security testing and quality assurance*. Artech House.

[68] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. 2017. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 2753–2762. https://proceedings.neurips.cc/paper/2017/hash/3a20f62a0af1aa152670bab3c602feed-Abstract.html

[69] Martin Tappler, Filip Cano Córdoba, Bernhard K. Aichernig, and Bettina Könighofer. 2022. Search-Based Testing of Reinforcement Learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, Luc De Raedt (Ed.). ijcai.org, 503–510. https://doi.org/10.24963/ijcai.2022/72

[70] CARLA Team. 2021. CARLA Challenge. https://carlachallenge.org/. Accessed on May 6, 2023.

[71] OpenAI Team. 2021. rlbaselines3-zoo. https://github.com/DLR-RM/rlbaselines3-zoo. Accessed on May 6, 2023.

[72] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[73] Miller Trujillo, Mario Linares-Vásquez, Camilo Escobar-Velásquez, Ivana Dusparic, and Nicolás Cardozo. 2020. Does Neuron Coverage Matter for Deep Reinforcement Learning?: A Preliminary Study. In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*. ACM, 215–220. https://doi.org/10.1145/3387940.3391462

[74] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1599–1614. https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi

[75] Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M Bayen. 2017. Flow: Architecture and benchmarking for reinforcement learning in traffic control. *arXiv preprint arXiv:1710.05465* 10 (2017).

[76] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. 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*. 146–157.

[77] Wen Xu, Hyungon Moon, Sanidhya Kashyap, Po-Ning Tseng, and Taesoo Kim. 2019. Fuzzing file systems via two-dimensional input space exploration. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 818–834.

[78] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. 2022. Revisiting Neuron Coverage Metrics and Quality of Deep Neural Networks. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 408–419. https://doi.org/10.1109/SANER53432.2022.00056

[79] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, Bowen Xu, Xin Zhou, DongGyun Han, and David Lo. 2023. Prioritizing Speech Test Cases. https://doi.org/10.48550/ARXIV.2302.00330

[80] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural Attack for Pre-Trained Models of Code. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1482–1493. https://doi.org/10.1145/3510003.3510146

[81] Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, et al. 2020. Towards playing full moba games with deep reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 621–632.

[82] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. 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* (Montpellier, France) *(ASE '18)*. Association for Computing Machinery, New York, NY, USA, 132–142. https://doi.org/10.1145/3238147.3238187

[83] Zhenya Zhang, Deyun Lyu, Paolo Arcaini, Lei Ma, Ichiro Hasuo, and Jianjun Zhao. 2023. FalsifAI: Falsification of AI-Enabled Hybrid Control Systems Guided by Time-Aware Coverage Criteria. *IEEE Trans. Software Eng.* 49, 4 (2023), 1842–1859. https://doi.org/10.1109/TSE.2022.3194640

[84] Ziyuan Zhong, Gail Kaiser, and Baishakhi Ray. 2022. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* (2022).

[85] Xin Zhou, Kisub Kim, Bowen Xu, DongGyun Han, Junda He, and David Lo. 2023. Generation-based Code Review Automation: How Far Are We? *arXiv preprint arXiv:2303.07221* (2023).

[86] Li ZHUO, Xiongfei WU, Derui ZHU, Mingfei CHENG, Siyuan CHEN, Fuyuan ZHANG, Xiaofei XIE, Lei MA, and Jianjun ZHAO. 2023. Generative model-based testing on decision-making policies. ASE.

[87] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. DeepHyperion: Exploring the Feature Space of Deep Learning-Based Systems through Illumination Search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 79–90. https://doi.

org/10.1145/3460319.3464811
[88] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. 2023. A Search-Based Testing Approach for Deep Reinforcement Learning Agents. *IEEE Transactions on Software Engineering* (2023), 1–22. https://doi.org/10.1109/TSE.2023.3269804