

Learning from the Test: Self-Referential Differential Testing for Deep RL Agents

JUNDA HE, Singapore Management University, Singapore

JIEKE SHI, Singapore Management University, Singapore

ZHOU YANG, University of Alberta, Canada and Canada CIFAR AI Chair, Amii, Canada

MINGFEI CHENG, Singapore Management University, Singapore

DAVID LO, Singapore Management University, Singapore

Deep Reinforcement Learning (DRL) has achieved significant success in complex decision-making problems. As DRL systems are increasingly deployed in real-world applications, ensuring their quality and reliability is paramount. Current works primarily focus on detecting safety-critical failures, often neglecting policy optimality, which can lead to reduced efficiency, user distrust, and economic losses. This oversight, compounded by the inherent “testing oracle problem” for optimality, leaves a significant gap in comprehensively evaluating DRL systems. To address this gap, we propose DELTA (Differential Testing for DRL Agents), a novel and comprehensive framework that automatically identifies both safety-critical and optimality bugs in DRL agents. DELTA employs a two-phase approach: (1) Safety Testing, where the Agent Under Test (AUT) is evaluated for catastrophic failures while collecting data from its decision-making policy, and (2) Optimality Testing, where this collected data from the prior phase is used to train a *challenger* agent via Offline Reinforcement Learning. Differential testing is then performed by comparing the challenger agent against the AUT; instances where the challenger achieves higher cumulative rewards indicate optimality issues in the AUT. We demonstrate DELTA’s effectiveness across five environments. We investigate the effectiveness of three offline RL algorithms (BC, BCQ, and CQL) in generating challenger agents. Experimental results demonstrate that safety testing datasets are valuable for training competent DRL agents. Challenger agents trained with BCQ proved most effective for identifying optimality issues within the framework of DELTA. Across the five environments, DELTA uncovered an average of 2,518 optimality issues, outperforming the baseline methods by 50.2%.

CCS Concepts: • **Computing methodologies** → **Markov decision processes**; • **Software and its engineering** → **Search-based software engineering**.

Additional Key Words and Phrases: Deep Reinforcement Learning, Differential Testing, Fuzz Testing

ACM Reference Format:

Junda He, Jieke Shi, Zhou Yang, Mingfei Cheng, and David Lo. 2026. Learning from the Test: Self-Referential Differential Testing for Deep RL Agents. *Proc. ACM Softw. Eng.* 3, ISSTA, Article ISSTA011 (October 2026), 25 pages. <https://doi.org/10.1145/3832102>

1 Introduction

In recent years, Deep Reinforcement Learning (DRL) [1, 49, 76] has gained significant attention for its ability to tackle complex decision-making problems. By interacting with unknown environments through trial-and-error, DRL agents learn a decision-making policy to maximize cumulative

Authors’ Contact Information: Junda He, jundahe.2022@phdcs.smu.edu.sg, Singapore Management University, Singapore, Singapore; Jieke Shi, jiekeshi@smu.edu.sg, Singapore Management University, Singapore, Singapore; Zhou Yang, zy25@ualberta.ca, University of Alberta, Edmonton, Canada and Canada CIFAR AI Chair, Amii, Edmonton, Canada; Mingfei Cheng, mfcheng.2022@phdcs.smu.edu.sg, Singapore Management University, Singapore, Singapore; David Lo, davidlo@smu.edu.sg, Singapore Management University, Singapore, Singapore.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2994-970X/2026/10-ARTISSTA011

<https://doi.org/10.1145/3832102>

reward over a sequence of actions. DRL has demonstrated remarkable success in various domains such as healthcare [8, 82], autonomous driving [29, 58, 60], and robotic manipulation [27, 31, 52], outperforming traditional control methods by a substantial margin [20, 46].

As DRL systems, and AI-based software systems more broadly [25], are increasingly deployed in real-world applications [53, 68], ensuring their quality and reliability becomes paramount [81]. Recent testing methodologies [26, 38, 55, 62] for DRL systems primarily focus on detecting catastrophic failures (safety-critical issues), e.g., a DRL-controlled robot colliding with obstacles. We refer to these testing methods as *safety testing*. Despite this progress, current safety testing methods largely overlook a critical dimension: the decision optimality of DRL systems. Evaluating policy optimality is as crucial as ensuring safety. For instance, a robotic arm performs a task with unnecessary energy consumption. Such non-optimal DRL decisions can incur higher economic costs, reduce operational efficiency, and decrease user satisfaction. More critically, unresolved optimality issues can increase the risk of escalating into safety-critical hazards; for example, a delivery robot selecting congested routes is more likely to face traffic accidents or battery depletion. True system reliability, therefore, demands both safety and optimality. To bridge the critical gap between safety and optimality evaluation, this paper presents a unified framework for testing DRL agents on detecting both *safety-critical failures* and *optimality issues*.

However, a fundamental challenge in testing the optimality of DRL policies is the absence of a clear testing oracle. DRL is predominantly applied in complex and dynamic environments. In these environments, the optimal solution that achieves the theoretical maximum cumulative reward is either unknown or analytically intractable. Without a ground-truth optimal solution to serve as a benchmark, it becomes difficult to assess the optimality of an agent's decisions quantitatively. This is akin to verifying the output of a software program without knowing the correct output. While manual assessment by domain experts is an option, it is prohibitively time-consuming and expertise-intensive, rendering it infeasible to scale for the vast and diverse scenarios DRL agents encounter. Automated, learned surrogates are therefore increasingly explored as scalable evaluators when a ground-truth oracle is unavailable, e.g., for judging software artifacts [24] and assessing the correctness of automatically generated patches [84]. The impracticality of manual assessment underscores the critical challenge of addressing the oracle problem and developing automated testing methods for DRL policy optimality.

To overcome this, differential testing [15, 47] emerges as a promising solution. This widely used software testing technique compares the outputs of multiple implementations of the software, thereby eliminating the need to know the correct output. Similarly, we adapt this principle to DRL. For a given DRL agent to be tested—we refer to this agent as the agent under test (AUT)—by comparing the AUT's cumulative reward against other DRL instances, we can reveal the AUT's optimality issues if it is outperformed in cumulative reward. However, this approach introduces a new challenge: generating suitable DRL instances for comparison. We can repeat the training process several times to obtain multiple DRL agents for differential testing; however, developers are typically only interested in testing high-performing DRL agents. Training such policies requires extensive interaction with the environment, consuming considerable computational resources and substantial time [11, 12]. For large-scale systems, restarting this process can take weeks or even months, requiring the processing of billions of data points [12].

To address this limitation, we introduce a more efficient approach founded on a core insight: the detailed records of an AUT's decision-making process generated during safety testing are a vibrant and underutilized resource for training highly competent DRL agents. The reasons are: (1) existing testing methodologies [26, 38, 55] employ various mechanisms to generate a broad spectrum of scenarios for evaluating the AUT and triggering diverse agent behaviors. These scenarios often include edge cases that are rarely encountered during standard training, deliberately pushing

the AUT into less explored regions. (2) Developers are typically only interested in testing high-performing DRL agents. Consequently, the AUT is a well-trained agent. Its behaviors, therefore, constitute a high-quality dataset of successful decisions. Leveraging this data provides a new policy with a strong starting point, jump-starts its training process, and bypasses the need for extensive, and often inefficient, initial exploration from scratch. (3) Finally, the resulting dataset offers a mixture of learning signals, which is crucial for robust training. It contains not only the AUT's successful trajectories as strong positive examples but also its documented failures as critical negative examples. This mixture provides clear guidance, teaching a new agent what behaviors to replicate and which to avoid.

As a result, we train a new agent using the AUT's own interaction data captured during safety testing. We name this new agent the *challenger* agent. Because it is trained solely on the AUT's own history, the challenger agent then serves as a potent self-referential testing oracle for suboptimality detection. This approach pragmatically relaxes the oracle problem. Acknowledging that true optimal policies are often unknown, if the challenger agent finds a superior solution, it exposes a concrete instance where the AUT's policy can be improved. As a result, this method effectively enables automated optimality testing without requiring a predefined perfect solution.

Notably, this approach yields value regardless of the testing outcome. A superior challenger identifies optimality issues and guides policy improvement. Conversely, a challenger that fails to outperform the AUT validates that the AUT is already near-optimal—without ever knowing what the true optimum is.

In this paper, we introduce DELTA (**D**ifferential **T**esting for DRL **A**gents), a comprehensive framework designed to automatically test DRL agents. DELTA seamlessly integrates established safety testing methods, such as CureFuzz [26], MDPFuzz [55], and GMT [38], with our proposed differential testing method, enabling holistic evaluation of agent quality in terms of both catastrophic failures and suboptimal behavior. DELTA consists of two main phases: **(1) Safety Testing:** The AUT is evaluated across a diverse set of scenarios to identify safety-critical issues. We adopt the state-of-the-art technique CureFuzz [26] for this phase. Simultaneously, DELTA records the AUT's interaction trajectories, creating a rich dataset that underpins subsequent optimality analysis. **(2) Optimality Testing:** Using the collected trajectories, DELTA applies Offline Reinforcement Learning [32, 36] to train a *challenger* agent. We then perform differential testing by comparing the challenger's performance against the original AUT. Cases where the challenger achieves higher cumulative rewards are flagged as optimality issues in the AUT.

To assess DELTA's effectiveness, we conducted evaluations across five diverse environments: three classical control tasks (CartPole, MountainCar, Acrobot) and two MuJoCo locomotion tasks (Hopper, Walker2D). We also evaluated the performance of three offline RL algorithms (BC [75], BCQ [21], and CQL [34]) in generating challenger agents. Experimental results demonstrate that safety testing datasets are valuable for training competent DRL agents. Challenger agents trained with BCQ proved most effective for identifying optimality issues within DELTA. Leveraging the BCQ-trained challenger agents, DELTA uncovered an average of 2,518 optimality issues per environment. DELTA significantly outperforms the baseline methods and identifies an average of 50.2% more optimality bugs across all environments.

The contributions of this paper include:

- We propose a novel self-referential testing oracle for detecting optimality bugs in DRL agents.
- We develop DELTA, a unified two-stage framework comprising a safety testing stage (which adopts state-of-the-art method CureFuzz) and our novel optimality testing stage, enabling DRL agents to be evaluated for both catastrophic failures and suboptimal decisions in a single pipeline.

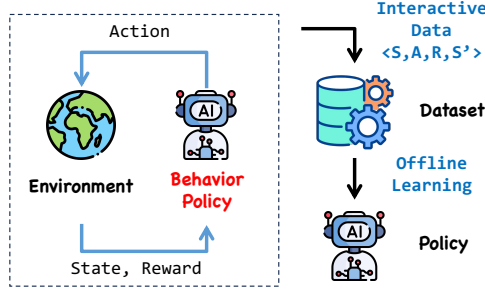


Fig. 1. Illustration of Offline Reinforcement Learning.

- We provide extensive experimental evidence demonstrating that DELTA effectively detects a significant number of optimality issues across diverse environments and DRL algorithms.

2 Preliminaries

2.1 Markov Decision Process

In general, reinforcement learning addresses the problem of learning to control a dynamical system [67]. Many RL problems are formalized using **Markov Decision Processes (MDPs)** [56], which are defined by a tuple $\langle S, S_0, A, T, R, \gamma \rangle$ where:

- S is a set of states. A state $s \in S$ represents a specific situation the agent encounters within the environment.
- $S_0 \subseteq S$ is the set of possible starting states. An initial state $s_0 \in S_0$ is selected for each episode.
- A is a set of actions. An action $a \in A$ is a decision made by the agent that affects the current state.
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition probability function, where $T(s'|s, a)$ represents the probability of transitioning from state s to state s' by taking action a .
- $R : S \times A \rightarrow \mathbb{R}$ is the reward function, where $R(s, a)$ is the immediate reward received for taking action a in state s . Typically, the agent's objective is to maximize the cumulative reward over time.
- $\gamma \in (0, 1]$ is a scalar discount factor that determines the present value of future rewards.

Markov Property. The Markov property implies that the future state depends solely on the current state and the action taken without being influenced by the sequence of past events. In this paper, we assume that the environments used in our experiments adhere to the Markov property.

Trajectory. A trajectory is a sequence of (state, action, reward) tuples.

Episode. An episode is always a trajectory that represents a complete run from a start state to a terminal state.

Policy. A policy $\pi : S \rightarrow A$ is a strategy that specifies the action a to be taken when in state s . The goal is to discover an optimal policy π^* that maximizes the expected cumulative reward over time. This objective can be formalized as the maximization of the expected sum of discounted rewards:

$$\max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{s_t, a_t \sim \pi} [\gamma^t R(s_t, a_t)] \quad (1)$$

where $R(s_t, a_t)$ represents the reward received at time step t for taking action a_t in state s_t , and $\mathbb{E}$ denotes the expectation over the states and actions under policy π .

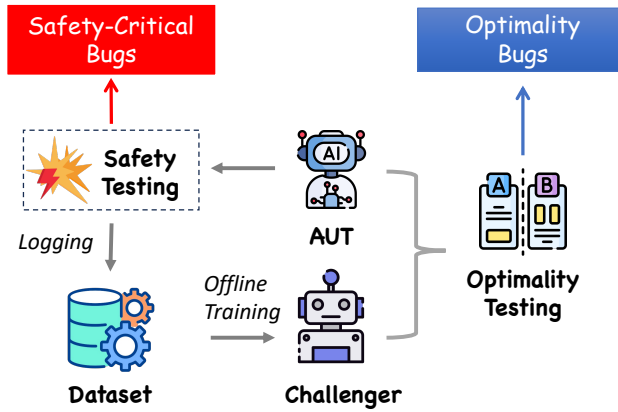


Fig. 2. The Overall Workflow of DELTA.

2.2 Offline Reinforcement Learning

Offline Reinforcement Learning (Offline RL) [32, 36] is an emerging paradigm in RL research. Unlike traditional RL, where agents learn by continuously interacting with the environment, offline RL focuses on creating effective policies exclusively from pre-collected datasets of trajectories without needing further environmental interaction (See Figure 1 as an example). This approach is particularly valuable in scenarios where interaction with the environment is expensive, time-consuming, or poses safety risks, such as in healthcare [70, 71], robotics [33, 64], or autonomous driving [16, 63]. Formally, in the offline RL setting [32, 36], learning occurs from a static dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$. The objective is to learn a policy $\pi : S \rightarrow A$ using only $\mathcal{D}$ that maximizes the expected cumulative discounted reward. The dataset $\mathcal{D}$ is generated by an existing policy (or set of policies), which is termed the behavior policy [19, 65]. This behavior policy can encompass a wide range of data collection strategies, from random exploration or rule-based approaches to expert demonstrations or policies from previously trained agents.

3 Approach

3.1 Assumption

Our methodology assumes a black-box testing paradigm: we do not require access to the AUT's internal states or parameters. Observations are limited to the AUT's interactions with the environment (i.e., state-action-reward-next_state tuples) generated during test execution. Furthermore, we assume the AUT adheres to a deterministic policy and the environment exhibits deterministic transition dynamics. While the approach is designed for environments modeled as Markov Decision Processes [56], it does not necessitate explicit knowledge of the environment's transition dynamics. This characteristic renders the method applicable to scenarios involving opaque AUTs, such as proprietary DRL agents or third-party services.

3.2 Overview

DELTA aims to detect both catastrophic failures and optimality bugs in DRL agents. As illustrated in Figure 2, it consists of two main phases:

1) Safety Testing. This phase aims to uncover safety-critical issues of the AUT, such as crashes or some hard constraint violations. In our current implementation, we utilize CureFuzz [26], a state-of-the-art safety testing method for DRL agents. Notice that DELTA can be seamlessly integrated

Algorithm 1 Differential Fuzzing for Optimality Testing

Input: Agent Under Test (AUT), Challenger Agent (CA)
Initialize seed corpus $\mathcal{S}$; curiosity networks T (fixed), P (trainable)
while time budget not exhausted **do**
 Select seed σ from $\mathcal{S}$ by energy $E(\sigma)$ (Eq. 7); mutate to σ'
 Execute AUT and CA on σ' ; obtain trajectories $\tau'_{\text{aut}}, \tau'_{\text{ca}}$ and rewards $R_{\text{aut}}(\sigma'), R_{\text{ca}}(\sigma')$
 if $R_{\text{ca}}(\sigma') > R_{\text{aut}}(\sigma') + \delta$ **then**
 Report optimality bug
 end if
 Compute diversity $D(\sigma')$ over τ'_{aut} (Eq. 3)
 Update P to minimize $\|T(s) - P(s)\|^2$ over $s \in \tau'_{\text{aut}}$
 Compute regret $A(\sigma')$ (Eq. 4) and inconsistency $I(\sigma')$ (Eq. 6)
 if $D(\sigma') > \bar{D}_S$ **or** $A(\sigma') > 1$ **or** $I(\sigma') < \bar{I}_S$ **then**
 Add σ' to $\mathcal{S}$
 end if
 Update energy $E(\sigma)$ for all seeds in $\mathcal{S}$
end while

with any other safety testing frameworks. Concurrently, all AUT behaviors observed during this fuzzing process are collected and stored as a dataset $\mathcal{D}$.

2) Optimality Testing. In the second phase, the dataset $\mathcal{D}$ collected from the safety testing phase is used to train a “challenger” agent, utilizing offline RL algorithms. This challenger agent serves as the testing oracle, guiding a differential fuzzing process to detect the AUT’s optimality issues. An optimality bug is subsequently flagged in any scenario where the challenger agent achieves a higher cumulative reward than the AUT. We introduce these two phases in detail as follows.

3.3 Safety Testing

Several safety testing methods have recently been proposed to detect crashes and hard constraint violations in DRL agents [26, 38, 55]. For our safety testing, we did not propose a new safety testing method. In our implementation, we adopt the state-of-the-art testing method, CureFuzz [26]. CureFuzz is a black-box fuzz testing approach. CureFuzz employs a “curiosity mechanism,” inspired by Random Network Distillation [5] and a multi-objective seed selection technique to optimize the fuzzing process to uncover a broad spectrum of crash-triggering scenarios; for details, we refer to the original CureFuzz paper [26].

Dataset Construction. Concurrently with the safety testing process, all observed trajectories are collected to form a static dataset $\mathcal{D}$. Each trajectory τ' , initiated from a seed σ' (via its initial state s'_0), consists of a sequence of state-action-reward-next_state tuples: $\tau' = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{H-1}$, where H is the length of the episode. The dataset $\mathcal{D} = \{\tau'_i\}_{i=1}^N$ thus aggregates all such trajectories recorded during fuzzing; the transition tuples they contain form the static dataset used for offline RL training (Section 2).

3.4 Optimality Testing

3.4.1 Overview. Optimality testing mainly consists of two key stages. First, a *challenger* agent is trained via offline RL algorithms [35, 36, 63] on the dataset $\mathcal{D}$ collected during safety testing. Then, this challenger is employed within a differential fuzzing process to identify optimality bugs of the AUT.

3.4.2 Oracle Formulation. We define an optimality issue as a scenario where the AUT achieves lower cumulative rewards than the challenger agent from the same initial state. The oracle, therefore,

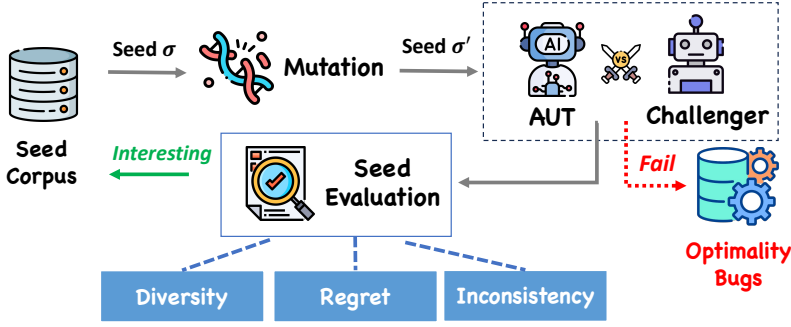


Fig. 3. DELTA's Differential Fuzzing Workflow.

relies on the reward function R , and its validity depends on what role R plays in our setting. We treat R as the reference objective for testing—the same objective the AUT was trained against. This follows standard DRL practice [67]. The design of R is a separate research problem, orthogonal to DELTA. That said, DELTA is naturally compatible with extended objectives. A practitioner can encode a new non-functional concern into R , and rerun DELTA against the extended objective.

Formally, the oracle condition for detecting an optimality issue is:

$$R(\tau_{ca}) > R(\tau_{aut}) + \delta \quad (2)$$

where τ_{aut} and τ_{ca} are the trajectories generated by the AUT and the challenger agent from the same initial state s_0 , $R(\tau)$ denotes the cumulative reward for trajectory τ , and $\delta \geq 0$ is a user-configurable magnitude threshold (default $\delta = 0$, recovering the strict inequality). All experiments in this paper use $\delta = 0$.

Challenger Agent Training. The dataset $\mathcal{D}$, gathered during the Safety Testing phase (Section 3.3), is utilized to train a challenger agent. We employ offline RL algorithms for this training, with specific algorithmic details provided in Section 4. The quality of the dataset $\mathcal{D}$ is critical to the challenger agent's performance, and the data gathered during safety testing is uniquely suitable for this purpose for several reasons.

First, safety testing naturally explores a wide spectrum of scenarios, including rare edge cases not encountered during standard training. This data diversity is crucial for training a challenger that can generalize across a broad range of situations [59, 66]. Second, the dataset contains a rich mixture of learning signals. By including both the AUT's successful trajectories and its documented failures, it provides clear positive and negative examples that guide the challenger on which behaviors to imitate and which to avoid [36, 54].

Finally, this approach is highly efficient. Since the dataset $\mathcal{D}$ is generated by a competent AUT, it provides a strong baseline that jump-starts the challenger's training, bypassing the need to learn from scratch. Moreover, the training is conducted entirely offline, requiring no additional, costly environment interactions. Another advantage of DELTA's design is the direct utilization of these existing safety testing logs. This ensures seamless integration with established safety testing and pragmatically eliminates the need for any separate, resource-intensive data collection phase specifically for training the challenger agent.

3.4.3 Differential Fuzzing. After training the challenger agent, DELTA employs differential fuzzing to identify optimality bugs. As depicted in Figure 3, this process leverages the challenger agent as a testing oracle to systematically evaluate the AUT's decisions.

Overview. DELTA follows an energy-guided, iterative fuzzing process. To begin, DELTA randomly generates an initial seed corpus, where each seed represents one initial state of the given environment. For example, in autonomous driving, a scenario can be described by the initial position of the autonomous vehicle and surrounding vehicles. In each iteration, DELTA selects a high energy seed from the corpus and mutates it slightly to generate a new seed (a new initial state). DELTA then observes both the AUT and the challenger’s actions under the new initial state to assess if the AUT performs suboptimally. An optimality bug is flagged if the challenger agent achieves a higher cumulative reward than the AUT. This cycle continues until the allocated time budget is exhausted. To guide the generation of diverse and compelling test cases, DELTA estimates the *energy* of the seed. Higher energy seeds, which indicate either novel, unexplored scenarios or greater potential to expose optimality bugs, are prioritized for mutation to produce new test cases. If the mutated seed is deemed “interesting,” it would be added back to the seed corpus for consideration in subsequent iterations.

Seed Energy Measurement. Seed energy is determined by three factors: *Diversity*, *Regret*, and *Inconsistency*. Hereafter, we denote a seed by σ . Each seed corresponds to an initial state, typically s_0 , from which the AUT generates a trajectory. We use s or s_t to represent a generic state within such a trajectory.

1) Diversity: Indicating a seed’s novelty compared to previously encountered seeds, diversity is key for exploring new AUT behaviors and scenarios. It is quantified using a curiosity mechanism [26]. The intuition behind this mechanism is to identify states that are “novel” to the fuzzer. It employs two neural networks: a *target network* T and a *predictor network* P . Both networks share the same architecture (e.g., a Multi-Layer Perceptron (MLP) [9]) and accept states encountered by the AUT as input. The target network T is initialized with random weights and remains fixed throughout the fuzzing process. The predictor network P , however, is continuously trained to mimic the output of T using the states observed during fuzzing.

Note that a randomly initialized DNN is fundamentally different from an unstructured random labeling. Although a DNN’s weights may be sampled randomly, the network itself is a deterministic, continuous function of its input: similar inputs produce similar outputs. This locality propagates to the predictor: once P has been trained to mimic T on a set of observed states, it also approximates T ’s output well for unseen states near them. The prediction error therefore stays low in well-explored regions and rises sharply in unexplored ones, which is precisely what we exploit as a novelty signal. Replacing T with per-state random labels would destroy this property: nearby inputs would carry uncorrelated targets, leaving P with no basis for extrapolation, and every unseen state would yield a similarly large error regardless of its proximity to observed ones.

Concretely, for an individual state s encountered in a trajectory, this novelty is measured by the Mean Squared Error [77] between the outputs of the two networks: $\|T(s) - P(s)\|^2$. The diversity of a seed σ (with initial state s_0), denoted $D(\sigma)$, is then calculated as the average of these novelty scores over all states in the trajectory τ generated by executing the AUT starting from s_0 :

$$D(\sigma) = \frac{1}{|\tau|} \sum_{s_t \in \tau} \|T(s_t) - P(s_t)\|^2 \quad (3)$$

2) Regret: Regret, denoted $A(\sigma)$, measures the performance difference between the AUT and the challenger agent for a specific initial state s_0 . It quantifies the AUT’s suboptimal performance relative to the challenger, reflecting the potential improvement had the AUT followed the challenger’s policy. A regret value $A(\sigma) > 1$ indicates that the challenger agent achieves a higher cumulative reward ($R^{\pi_{ca}}$) than the AUT ($R^{\pi_{aut}}$) for that scenario. Formally, $A(\sigma)$ is calculated as:

$$A(\sigma) = \exp(R^{\pi_{ca}}(s_0) - R^{\pi_{aut}}(s_0)) \quad (4)$$

This exponential formulation of regret serves to significantly amplify scenarios where the challenger agent markedly outperforms the AUT. By emphasizing this relative performance gain, $A(\sigma)$ guides the fuzzer towards test cases demonstrating clear potential for the AUT's policy improvement, rather than merely towards situations of poor AUT performance that might arise from inherently challenging or unsolvable aspects of the environment for both agents.

3) Inconsistency: The Inconsistency score, $I(\sigma)$, quantifies the behavioral overlap between the AUT and the challenger agent when starting from the same initial seed σ ; a lower score indicates greater behavioral divergence. It aims to identify scenarios where the two agents explore dissimilar state sequences, potentially revealing instances where the challenger's actions differ significantly from those of the AUT.

To measure inconsistency, we employ Locality Sensitive Hashing (LSH) [28]. LSH utilizes a hash function $h(s)$ to map high-dimensional states s to lower-dimensional hash codes. We implement LSH based on Euclidean distance [10]. The process can be conceptualized as follows:

- (a) **Random Hyperplane Generation:** A set of k random hyperplanes is defined within the high-dimensional state space. Each hyperplane j passes through the origin and is characterized by its random normal vector r_j .
- (b) **State Projection and Discretization:** For any given state s , its projection onto each random normal vector r_j is calculated as the dot product: $p_j(s) = s \cdot r_j$. The resulting projected value is then binarized based on its sign. Specifically, a single bit $g_j(s)$ is generated for each hyperplane, indicating on which side of the j -th hyperplane the state s lies:

$$g_j(s) = \begin{cases} 1 & \text{if } p_j(s) > 0 \\ 0 & \text{if } p_j(s) \leq 0 \end{cases}$$

- (c) **Binary Hash Code:** The resulting sequence of k binary digits, $(g_1(s), g_2(s), \dots, g_k(s))$, forms the discrete hash code, $h(s)$, for state s .

The key property is that if two states s_i and s_j are close in Euclidean distance, their projections onto most random lines are also likely to be close, causing them to fall into the same or nearby segments. This results in a high probability that their composite hash codes, $h(s_i)$ and $h(s_j)$, will be identical. Conversely, states far apart in Euclidean distance are more likely to be separated into different segments on many lines, yielding different hash codes.

For a given seed σ , let τ_{aut} and τ_{ca} be the trajectories of the AUT and the challenger agent, respectively. We apply the LSH function h to each state in these trajectories, yielding two sets of unique hash codes that represent the distinct regions explored by each agent:

$$H_{\text{aut}} = \{h(s_t) \mid s_t \in \tau_{\text{aut}}\}, \quad H_{\text{ca}} = \{h(s_t) \mid s_t \in \tau_{\text{ca}}\} \quad (5)$$

The inconsistency metric $I(\sigma)$ is calculated as the number of LSH hash codes commonly visited by both the AUT and the challenger agent, divided by the total number of unique hash codes visited by the AUT.

$$I(\sigma) = \frac{|H_{\text{aut}} \cap H_{\text{ca}}|}{|H_{\text{aut}}|} \quad (6)$$

This energy function $E(\sigma)$ for optimality testing is defined as:

$$E(\sigma) = \alpha \cdot D(\sigma) + \beta \cdot A(\sigma) + \lambda \cdot (1 - I(\sigma)) \quad (7)$$

where $D(\sigma)$ is the diversity score, $A(\sigma)$ is the regret (Equation 4), and $I(\sigma)$ is the inconsistency score (Equation 6). The terms α, β, λ are weighting factors for these components.

Mutation. Mutation is the process that generates a new test scenario (σ') from an existing seed (σ) by applying a slight random perturbation to the initial state vector represented by the seed. The nature of this perturbation is adapted to the specific initial state space of each environment.

For instance, in the CartPole environment, a seed's initial state is defined by a vector of four values: cart position, cart velocity, pole angle, and pole angular velocity. To mutate a CartPole seed, a small random value is added to each of these four components, creating a new, slightly different starting condition for the subsequent test run. The specific descriptions of the initial states for all environments are detailed in Section 4.

Fuzzing. Algorithm 1 provides a detailed description of the differential fuzzing. It mainly consists of two parts: *Initialization* and *Fuzzing Iteration*.

1) *Initialization.* DELTA generates the initial population of seeds $\mathcal{S}$ by randomly sampling the initial state of the environment based on a uniform distribution. Meanwhile, DELTA initializes the Curiosity module to assign the diversity of each seed in the seed corpus.

2) *Fuzzing Iteration.* The core fuzzing loop proceeds as follows for each iteration until the time budget is exhausted: A seed σ is selected from the corpus $\mathcal{S}$, where the probability of selecting any seed σ_i is proportional to its energy $E(\sigma_i)$ relative to the total energy of all seeds in the corpus:

$$P(\sigma_i) = \frac{E(\sigma_i)}{\sum_{\sigma_j \in \mathcal{S}} E(\sigma_j)} \quad (8)$$

The selected seed σ is then mutated by applying a slight random perturbation, yielding a new seed σ' . The AUT and challenger's behavior, when initiated from σ' (i.e., its initial state s'_0), is subsequently monitored.

Dynamic Thresholding. Manually setting fixed thresholds to identify “interesting” seeds for fuzzing can be challenging and often requires domain-specific expertise. Instead, DELTA utilizes a dynamic thresholding mechanism during its differential fuzzing phase to determine which new seeds σ' are added to the corpus $\mathcal{S}$. A seed σ' is considered interesting and added if it meets at least one of the following criteria:

- (1) Its **diversity** $D(\sigma')$ exceeds the average diversity $\bar{D}_{\mathcal{S}}$ of seeds currently in the corpus (i.e., $D(\sigma') > \bar{D}_{\mathcal{S}}$), indicating it explores novel states.
- (2) Its **regret** $A(\sigma')$ is greater than 1 (i.e., $A(\sigma') > 1$). This signifies that the challenger agent achieved a higher cumulative reward than the AUT for seed σ' , directly indicating an instance of AUT suboptimality.
- (3) Its **inconsistency** $I(\sigma')$, which measures trajectory overlap as defined in Equation 6, falls below the average inconsistency $\bar{I}_{\mathcal{S}}$ of seeds in the corpus (i.e., $I(\sigma') < \bar{I}_{\mathcal{S}}$). A lower $I(\sigma')$ value highlights the greater behavioral divergence between the AUT and the challenger agent.

Subsequently, at the end of each iteration, DELTA updates the energy values (calculated using Equation 7) for all seeds remaining in the corpus $\mathcal{S}$. This process of seed selection, mutation, evaluation, and corpus management continues until the allocated time budget is exhausted.

4 Experiment Setting

Environment To assess the feasibility of DELTA, we apply it to multiple case studies, including three classical control environments: CartPole, MountainCar, and Acrobot, as well as two MuJoCo environments: Hopper and Walker2D. These environments are widely used in previous work on testing DRL agents [41, 79, 85].

- **CartPole:** The agent balances a pole on a cart by moving left or right. Success is keeping it upright; episodes last up to 500 steps. Initial states vary by cart position, cart velocity, pole angle, and angular velocity.
- **MountainCar:** The objective is to navigate an underpowered car from a random starting position and zero velocity to a hilltop. The agent controls the car by applying directional forces. The agent receives a reward of -1 per step until success or 200 steps.

- **Acrobot:** The agent swings a two-link pendulum to a target height. It receives -1 reward per step until reaching the target or 500 steps. The initial state involves the cosine and sine values of the angles and their corresponding angular velocities.
- **Hopper:** A one-legged robot aims to hop forward by applying torques to its three actuated joints. Initial states include noise on joint angles and velocities. Rewards are based on forward velocity, survival, and penalties for excessive action. An episode terminates if the Hopper enters an “unhealthy” state or after 1000 timesteps.
- **Walker2D:** A bipedal robot learns to walk forward by applying torques to its six actuated joints. Initial states include slight random variations in joint angles and velocities. Similar to Hopper, rewards favor forward progress and upright posture, and episodes end upon entering an “unhealthy” state or after 1000 timesteps.

Safety Testing Oracle A safety violation or catastrophic failure is defined for each environment as follows: We define safety violations based on standard environment termination criteria [17, 18]. For *CartPole*, failure occurs if the pole tilts $> 12^\circ$ or the cart moves > 2.4 units. *MountainCar* fails if the goal is not reached in 200 steps. *Acrobot* is considered unsatisfactory if the reward drops below -100. For *Hopper* and *Walker2D*, a violation occurs if the robot becomes “unhealthy” (e.g., falling or exceeding joint limits).

AUT Implementation We train the AUT using stable and verified implementations of DRL algorithms from the *d3rlpy* open-source library [59]. Specifically, for environments with discrete action spaces (*CartPole*, *MountainCar*, and *Acrobot*), we employ the Double Deep Q-Network (DDQN) algorithm [76]. For environments with continuous action spaces (*Hopper* and *Walker2D*), we utilize the Soft Actor-Critic (SAC) algorithm [22].

We confirmed the high performance of all AUTs prior to safety testing. We report the average cumulative reward and 95% confidence interval margin of error over 100 runs for each environment, with detailed results presented in Table 1.

Challenger Implementation To train the challenger agent, we leverage offline Reinforcement Learning (RL) algorithms, a rapidly evolving subfield of RL. We experiment with three prominent offline RL methods:

- **BC** [75] directly learns a policy by mimicking state-action pairs from a static dataset.
- **BCQ** [21] mitigates distributional shift by constraining the policy to select actions similar to those in the provided data batch.
- **CQL** [34] addresses the overestimation of Q-values for out-of-distribution actions by learning a conservative Q-function; it incorporates a regularizer that minimizes Q-values for actions not well-represented in the dataset while maximizing them for dataset actions.

For the MuJoCo environments, we utilized hyperparameters from the D4RL replication package [19]. Since D4RL does not provide hyperparameters for the classical control environments, we adopted the hyperparameters from the Scope-RL replication package [30]. All experiments were implemented in Python 3.9 and conducted on an Ubuntu 22.04 server equipped with an AMD EPYC 7643 48-core Processor, 504 GB RAM, and four NVIDIA RTX A5000 GPUs.

5 Results

RQ1: Which offline RL algorithm is most effective for training challenger agents to identify optimality bugs?

Safety Testing. We first perform the safety testing phase to identify catastrophic failures in the AUT. Following common practice [23, 44, 83], each run is executed for two hours, which includes a 30-minute sampling phase to construct the initial seed corpus. We perform five independent runs of safety testing for each environment using distinct random seeds. Throughout the entire

Table 1. Number of safety failures detected in the AUT, AUT success rate, and mean cumulative reward over 100 episodes for the AUT and challengers (BC, BCQ, CQL). Brackets [lower, upper] denote the 95% Wilson score CI for the success rate; $\pm$ denotes the 95% CI margin of error for the other columns. Bold values indicate the best-performing challenger agent (among BC, BCQ, and CQL) for each environment.

Environment	Number of Safety Failures of AUT	AUT Success Rate (%)	AUT Performance	BC Performance	BCQ Performance	CQL Performance
CartPole	77.4 $\pm$ 10.3	97.0 [91.6, 99.0]	495.7 $\pm$ 6.8	497.1 $\pm$ 2.5	496.0 $\pm$ 2.6	499.8 $\pm$ 1.6
MountainCar	167.2 $\pm$ 8.7	99.0 [94.6, 99.8]	-129.8 $\pm$ 2.8	-129.0 $\pm$ 1.4	-128.2 $\pm$ 1.4	-126.6 $\pm$ 1.5
Acrobot	1183.6 $\pm$ 41.9	93.0 [86.3, 96.6]	-81.3 $\pm$ 3.0	-83.7 $\pm$ 2.1	-84.2 $\pm$ 3.2	-83.3 $\pm$ 1.5
Hopper	154.4 $\pm$ 9.9	97.0 [91.6, 99.0]	3162.3 $\pm$ 60.9	3173.4 $\pm$ 41.6	3204.4 $\pm$ 32.4	3173.8 $\pm$ 20.8
Walker2D	251.8 $\pm$ 23.8	95.0 [88.8, 97.9]	3925.5 $\pm$ 192.2	3961.3 $\pm$ 95.8	4004.0 $\pm$ 92.1	3996.5 $\pm$ 68.5

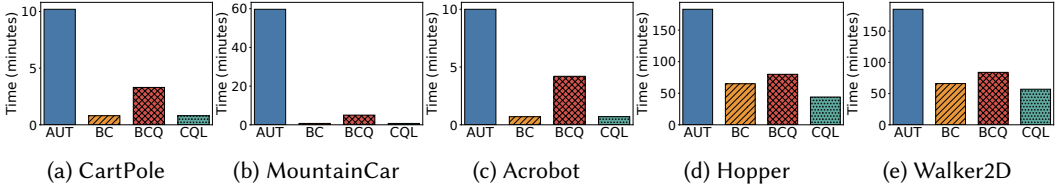


Fig. 4. Running time of training the Agent Under Test (AUT), and the challenger agents (BC, BCQ and CQL).

safety testing process, covering both the sampling and fuzzing phases, every trajectory executed by the AUT is recorded into the dataset $\mathcal{D}$ used for challenger training. Table 1 reports, for each environment, the average number of detected failures and the 95% confidence interval margin.

Performance of Challenger Agents. We detail our experimental setup for training challenger agents as follows:

- 1) *Dataset Generation*: Safety testing is conducted five times per environment, yielding five distinct datasets for each environment.
- 2) *Challenger Agent Training*: Each of these five datasets was utilized to train three challenger agents using different offline RL algorithms (BC, BCQ, and CQL). This approach yielded a total of 75 challenger agents (5 environments $\times$ 5 datasets $\times$ 3 algorithms).
- 3) *Evaluation Metrics*: We calculate the *Mean Cumulative Reward* of one DRL agent under 100 independent runs. The overall performance metrics for a specific offline RL algorithm in a given environment are calculated as the average of its five respective challenger agents. From Table 1, we can observe no single offline RL algorithm uniformly achieves the highest cumulative rewards across all environments; both CQL and BCQ frequently achieved comparable or superior performance. In general, the challenger agents perform very closely to the AUT. This comparable performance implies they are valuable and competent agents for conducting differential analysis with the AUT.

Efficiency of Training Challenger Agents. The requirement to train a challenger agent using offline RL may raise concerns about computational overhead. In Figure 4, we illustrate the training time of the AUT and challenger agents. Since there are five challenger agents trained for each offline RL algorithm in each environment, we report the median value. However, we notice that the training time is almost identical across runs. In general, the training time for the challenger agents is significantly lower than that of the AUT across all environments. This efficiency gain is mainly due to the different learning paradigms employed [78]. The AUT is trained using traditional reinforcement learning, which requires extensive and often time-consuming interaction with the environment to gather data and learn a policy. In contrast, the challenger agents learn exclusively from a pre-collected, static dataset of trajectories, eliminating the need for any further interaction with the environment.

Table 2. Experiment Results of RQ1. We compare the performance of different offline RL algorithms for training the challenger agents to detect optimality bugs. We report the Number of Bugs, the Number of Distinct Bugs, and the Distance. The average result over five runs is reported. The $\pm$ values represent the 95% confidence interval margin of error. Bold values indicate the best result in each column.

# of Bugs	CartPole	MountainCar	Acrobot	Hopper	Walker2D
BC	79.4 $\pm$ 8.5	1243.2 $\pm$ 318.3	2900.0 $\pm$ 797.3	1231.8 $\pm$ 236.8	1251.4 $\pm$ 148.9
BCQ	105.6 $\pm$ 17.7	3752.0 $\pm$ 649.9	5450.0 $\pm$ 1241.7	1738.8 $\pm$ 315.6	1543.2 $\pm$ 146.6
CQL	84.4 $\pm$ 10.7	2144.0 $\pm$ 797.6	4811.0 $\pm$ 760.1	875.0 $\pm$ 123.5	778.6 $\pm$ 75.7
# of Distinct Bugs	CartPole	MountainCar	Acrobot	Hopper	Walker2D
BC	65.4 $\pm$ 9.8	19.8 $\pm$ 6.9	298.2 $\pm$ 12.2	1183.6 $\pm$ 228.8	1238.4 $\pm$ 150.8
BCQ	81.0 $\pm$ 9.8	58.2 $\pm$ 9.9	467.4 $\pm$ 10.0	1648.8 $\pm$ 280.8	1508.2 $\pm$ 213.4
CQL	67.4 $\pm$ 8.5	23.8 $\pm$ 5.6	432.8 $\pm$ 15.4	852.8 $\pm$ 115.8	771.0 $\pm$ 75.8
Distance	CartPole	MountainCar	Acrobot	Hopper	Walker2D
BC	0.067 $\pm$ 3e-3	0.019 $\pm$ 8e-3	0.155 $\pm$ 3e-3	0.014 $\pm$ 2e-4	0.018 $\pm$ 2e-4
BCQ	0.071 $\pm$ 3e-3	0.060 $\pm$ 2e-2	0.157 $\pm$ 3e-3	0.014 $\pm$ 2e-4	0.019 $\pm$ 2e-4
CQL	0.068 $\pm$ 2e-3	0.020 $\pm$ 8e-3	0.156 $\pm$ 3e-3	0.014 $\pm$ 2e-4	0.017 $\pm$ 2e-4

The notably long AUT training time in MountainCar stems from the environment's challenging reward system. In MountainCar, an agent is penalized at every timestep (reward of -1) and only receives a positive reward upon reaching the hilltop goal. This lack of any intermediate positive feedback creates a major exploration challenge: at the beginning of the training, all actions appear equally poor to the agent. It cannot learn incrementally and must instead discover a rare, successful trajectory through prolonged random exploration, which accounts for the significant training time observed.

Optimality Testing. Further, we evaluate the effectiveness of the different challenger agents (BC, BCQ, and CQL) in identifying optimality bugs. We conduct five differential fuzzing executions for every combination of environment and offline RL algorithm. In each of these five executions, the AUT is compared against one of the five previously generated challenger agents. Following the same setup as safety testing, each optimality testing execution had a duration of two hours. We focus on the following evaluation metrics:

- **# of Bugs:** The total count of optimality issues.
- **# of Distinct Bugs:** The space of the initial state is discretized into a grid by dividing each dimension into a predefined number of intervals. We use five intervals, following previous work [45], except for the MountainCar environment, where 100 intervals are employed due to its simpler nature. The number of Distinct Bugs is then the number of unique grid cells containing at least one detected bug.
- **Distance:** Assesses the sparseness of the detected optimality issues within the state space. It is calculated as the average Euclidean distance over all pairs of scenarios where optimality issues were identified.

Results. For all evaluation metrics, we report the average performance over the five fuzzing executions and the margin of error at 95% confidence interval. Table 2 demonstrates the capability of BC, BCQ, and CQL in training challenger agents to detect optimality bugs. Across all five environments, the BCQ-trained challenger agents consistently proved to be the most effective at identifying optimality bugs. They detected the highest *Number of Bugs* and *Number of Distinct Bugs* in every environment. For instance, in the MountainCar environment, the BCQ challenger found an average of 3752.0 bugs, significantly outperforming both BC (1243.2) and CQL (2144.0).

BCQ-trained challenger agents are therefore established as the default choice to form the oracle when implementing DELTA.

An interesting insight arises when comparing these bug detection results (Table 2) with the performance of the challenger agents (Table 1). In cases where CQL-trained challenger agents achieved the highest cumulative rewards, they were less effective at optimality bug detection than BCQ agents. This divergence suggests that the agent characteristics that lead to the highest task rewards are not necessarily the same as those that are most effective for uncovering optimality issues under DELTA.

Answer to RQ1: BCQ-trained challenger agents are the most effective for DELTA in detecting optimality bugs, yielding an average of 2,517.9 issues per environment, and are therefore established as the default choice when implementing DELTA.

RQ2: How effective is DELTA at detecting optimality bugs compared to baseline methods?

Baseline Methods. To the best of our knowledge, DELTA is the first work on this topic and there are no direct baselines. Metamorphic testing (e.g., Decictor [6]) relies on manually-defined domain-specific metamorphic relations. General ML testers (e.g., DeepGauge [43]) target static problems. Consequently, we select three state-of-the-art safety testing methods. These methods also prioritize exploration diversity, ensuring broad coverage of the state space.

- CureFuzz [26]: As we described in Section 3, CureFuzz is the state-of-the-art fuzz testing approach to uncover crashes of DRL agents.
- MDPFuzz [55]: A black-box fuzz testing framework for deep learning models that solve problems modeled as Markov Decision Processes.
- GMT [38]: A framework that utilizes a generative diffusion model to create test cases and uses novelty-based guidance to diversify agent behaviors.

For a fair and direct comparison, all testing methodologies operated under identical conditions: they used the same BCQ-trained challenger agent (from RQ1) to form the differential testing oracle and were allotted a two-hour time budget. To ensure the statistical reliability of our findings, we repeated each experiment five times and report the average performance.

Results. The experimental results are presented in Table 3. To summarize, DELTA achieves the best overall performance when compared against the baseline methods across all environments and nearly all evaluation metrics. On average, DELTA identifies 50.2% more optimality bugs across all environments. Statistical analysis using the Mann-Whitney U test [48] and Cohen's d effect size [7] confirms that DELTA's superior performance in finding both the total number of bugs and distinct bugs is statistically significant and substantial. Regarding the total Number of Bugs (# of Bugs), DELTA outperformed all baseline methods in all five environments, consistently identifying the highest total number of optimality issues. This highlights its effectiveness in locating instances of suboptimal agent behavior. The most significant advantage of DELTA is its ability to find a diverse set of unique bugs. It found the highest number of distinct bugs (# of Distinct Bugs) in every environment, often by a large margin. For example, in the Hopper environment, DELTA found more than double the number of distinct bugs compared to the closest competitor (1648.8 vs. 769.2). In the Acrobot environment, DELTA identifies over five times more distinct bugs than the runner-up (467.4 vs. 93.2).

On the Distance metric, which measures the spread of the discovered bugs, DELTA's performance is competitive but not universally dominant. It achieved the highest or joint-highest distance score in four of the five environments (MountainCar, Acrobot, Hopper, and Walker2D). The only

Table 3. Performance comparison of DELTA against baseline methods (MDPFuzz, CureFuzz, and GMT) in detecting optimality bugs, including the total number of bugs found (# of Bugs), the number of unique bugs (# of Distinct Bugs), and Distance. All results are averaged over five runs, with the $\pm$ values representing the 95% confidence interval margin of error. Bold values indicate the best performance in each row. The percentages in parentheses for DELTA denote its improvement over the best-performing baseline for that metric.

Environment	# of Bugs			
	DELTA	MDPFuzz	CureFuzz	GMT
CartPole	105.6 $\pm$ 17.7 (+46.7%)	54.0 $\pm$ 17.6	72.0 $\pm$ 8.1	60.6 $\pm$ 15.5
MountainCar	3752.0 $\pm$ 649.9 (+91.5%)	1854.4 $\pm$ 600.6	1959.2 $\pm$ 383.6	739.8 $\pm$ 559.1
Acrobot	5450.0 $\pm$ 1241.7 (+40.9%)	3869.0 $\pm$ 1425.4	2556.6 $\pm$ 861.9	1850.8 $\pm$ 598.5
Hopper	1738.8 $\pm$ 315.6 (+37.9%)	674.2 $\pm$ 101.3	815.0 $\pm$ 160.1	1261.2 $\pm$ 286.2
Walker2D	1543.2 $\pm$ 146.6(+34.1%)	598.6 $\pm$ 141.1	838.8 $\pm$ 342.7	1150.4 $\pm$ 193.9
Environment	# of Distinct Bugs			
	DELTA	MDPFuzz	CureFuzz	GMT
CartPole	81.0 $\pm$ 9.8 (+71.6%)	43.4 $\pm$ 8.2	47.2 $\pm$ 3.9	43.6 $\pm$ 7.9
MountainCar	58.2 $\pm$ 9.9 (+19.3%)	48.8 $\pm$ 6.7	47.8 $\pm$ 10.3	35.6 $\pm$ 4.7
Acrobot	467.4 $\pm$ 10.0 (+401.5%)	93.2 $\pm$ 14.9	75.8 $\pm$ 9.3	63.6 $\pm$ 5.2
Hopper	1648.8 $\pm$ 280.8 (+114.4%)	641.8 $\pm$ 103.1	769.2 $\pm$ 149.1	616.4 $\pm$ 124.7
Walker2D	1508.2 $\pm$ 213.4 (+81.8%)	592.4 $\pm$ 140.0	829.6 $\pm$ 339.1	487.2 $\pm$ 128.7
Environment	Distance			
	DELTA	MDPFuzz	CureFuzz	GMT
CartPole	0.071 $\pm$ 3e-3	0.070 $\pm$ 3e-3	0.070 $\pm$ 3e-3	0.084 $\pm$ 3e-3
MountainCar	0.060 $\pm$ 2e-2	0.052 $\pm$ 1e-2	0.060 $\pm$ 2e-2	0.025 $\pm$ 5e-3
Acrobot	0.157 $\pm$ 3e-3 (+61.9%)	0.071 $\pm$ 3e-3	0.068 $\pm$ 2e-3	0.097 $\pm$ 5e-3
Hopper	0.014 $\pm$ 2e-4	0.014 $\pm$ 1e-4	0.014 $\pm$ 1e-4	0.010 $\pm$ 8e-4
Walker2D	0.019 $\pm$ 2e-4 (+11.8%)	0.017 $\pm$ 1e-4	0.017 $\pm$ 2e-4	0.014 $\pm$ 3e-4

environment where a baseline method achieved a distinctly better result was CartPole. In CartPole, the generative approach of GMT found fewer distinct bugs than DELTA (43.6 vs. 81.0), yet it achieved the highest distance score (0.084 vs. 0.071). This implies that the few bugs GMT finds are likely to be far apart and unrelated. In contrast, DELTA finds a dense cluster of many unique bugs that are, on average, closer together.

In summary, while not leading in every single metric, the consistent dominance of DELTA in discovering the most total and the most unique bugs makes it the most effective testing framework overall.

Answer to RQ2: DELTA significantly outperforms the baseline methods in terms of nearly all evaluation metrics in optimality bug detection. On average, DELTA identifies 50.2% more optimality bugs.

RQ3: To what extent does each component of the energy function (i.e., diversity, regret, and inconsistency) contribute to DELTA's optimality bug detection performance?

To investigate this research question, we implement and experiment with three ablated variants of DELTA (w/o diversity, w/o regret, w/o inconsistency). Table 4 summarizes the experimental results. The removal of the Regret component (*w/o-Regret*) causes the most substantial drop in the total number of bugs found. For example, in the Acrobot environment, the bug count falls by approximately 59% from 5450.0 to 2238.4. This pattern confirms that prioritizing test cases where the challenger agent shows a clear performance advantage is the single most effective strategy for finding a high volume of optimality bugs. The Diversity component is crucial for discovering a wide range of unique bugs. Its removal (*w/o-Diversity*) leads to a dramatic reduction in the number of Distinct Bugs. This is most clearly seen in the CartPole environment, where the number of

Table 4. Performance comparison of DELTA against ablated variants (w/o diversity, w/o regret, w/o inconsistency) in detecting optimality bugs, including the total number of bugs found (# of Bugs), the number of unique bugs (# of Distinct Bugs), and Distance. All results are averaged over five runs, with the $\pm$ values representing the 95% confidence interval margin of error. Bold values indicate the best performance in each column.

Method	# of Bugs				
	CartPole	MountainCar	Acrobot	Hopper	Walker2D
DELTA	105.6$\pm$17.7	3752.0$\pm$649.9	5450.0$\pm$1241.7	1738.8$\pm$315.6	1543.2$\pm$146.6
w/o-Diversity	93.8 $\pm$ 7.7	2373.0 $\pm$ 269.4	2963.0 $\pm$ 446.4	1464.6 $\pm$ 271.6	1252.0 $\pm$ 311.7
w/o-Regret	74.6 $\pm$ 11.3	2328.0 $\pm$ 252.6	2238.4 $\pm$ 165.9	1409.8 $\pm$ 214.1	1009.4 $\pm$ 247.9
w/o-Inconsistency	67.6 $\pm$ 9.0	2509.2 $\pm$ 526.3	2532.4 $\pm$ 295.2	1419.4 $\pm$ 350.9	1147.2 $\pm$ 206.0
Method	# of Distinct Bugs				
	CartPole	MountainCar	Acrobot	Hopper	Walker2D
DELTA	81.0$\pm$9.8	58.2$\pm$9.9	467.4$\pm$10.0	1648.8$\pm$280.8	1508.2$\pm$213.4
w/o-Diversity	39.4 $\pm$ 2.4	43.8 $\pm$ 6.4	343.2 $\pm$ 38.8	1329.4 $\pm$ 197.9	1235.8 $\pm$ 308.7
w/o-Regret	63.0 $\pm$ 8.9	48.0 $\pm$ 10.5	409.4 $\pm$ 17.2	1299.0 $\pm$ 280.8	895.6 $\pm$ 240.2
w/o-Inconsistency	60.0 $\pm$ 7.7	47.6 $\pm$ 10.2	448.4 $\pm$ 16.7	1315.8 $\pm$ 212.3	1037.4 $\pm$ 155.7
Method	Distance				
	CartPole	MountainCar	Acrobot	Hopper	Walker2D
DELTA	0.071$\pm$3e-3	0.060 $\pm$ 2e-2	0.157 $\pm$ 3e-3	0.014 $\pm$ 2e-4	0.019 $\pm$ 2e-4
w/o-Diversity	0.070 $\pm$ 1e-3	0.057 $\pm$ 2e-2	0.157 $\pm$ 3e-3	0.014 $\pm$ 1e-4	0.018 $\pm$ 2e-4
w/o-Regret	0.069 $\pm$ 2e-3	0.061$\pm$2e-2	0.158$\pm$3e-3	0.014 $\pm$ 1e-4	0.019 $\pm$ 2e-4
w/o-Inconsistency	0.070 $\pm$ 2e-3	0.060 $\pm$ 2e-3	0.157 $\pm$ 3e-3	0.014 $\pm$ 1e-4	0.018 $\pm$ 2e-4

distinct bugs is more than halved, dropping from 81.0 to 39.4. Further, removing Inconsistency (w/o-Inconsistency) also results in a noticeable decrease in both total and distinct bugs across all environments.

The ablation study concludes that all three energy components, i.e., Regret, Diversity, and Inconsistency, are integral to the success of DELTA. They all play synergistic and distinct roles. Regret is the primary factor for the quantity of bugs.

Answer to RQ3: The ablation study demonstrates that Diversity, Regret, and Inconsistency are all vital for DELTA's performance. Regret is the single most crucial factor for maximizing the total number of optimality bugs found.

RQ4: How useful are the identified optimality bugs to improve the performance of DRL agents?

We investigate whether the optimality bugs identified by DELTA can be leveraged to enhance the performance of the AUT. The core strategy involves fine-tuning the AUT using the superior trajectories generated by the challenger agent during the optimality testing phase. The experimental methodology is structured into the following stages:

- (1) *Curation of a Superior Trajectory Dataset:* First, based on the identified optimality bugs, we create a new dataset, $\mathcal{D}_{\text{superior}}$. This dataset is formed by collecting every trajectory where the challenger agent outperformed the AUT. This dataset is thus composed entirely of high-performing trajectories that demonstrate policies superior to the AUT's original behavior.
- (2) *Offline Fine-Tuning:* The original AUTs are further trained using this new dataset. Notice that the AUT cannot be effectively fine-tuned on the static dataset $\mathcal{D}_{\text{superior}}$ using its original online training algorithm (e.g., DDQN or SAC), as these methods perform poorly in offline settings [16, 34, 35]. Therefore, our approach is to retain the AUT's existing neural network architecture but switch its training objective to one specifically designed for the offline paradigm. We utilize Conservative

Table 5. Comparison of the original Agent Under Test (AUT) and the AUT-Improved. The first two columns report the mean cumulative reward for each agent. The third and fourth columns present the average number of safety failures on known safety-critical scenarios, with percentages in parentheses quantifying the failure reduction after repair. The last column reports the results of the re-evaluation of CureFuzz against the AUT-Improved. All $\pm$ values represent the 95% confidence interval margin of error.

Environment	Mean Reward (AUT)	Mean Reward (AUT-Improved)	Safety Failures of AUT	Safety Failures of AUT-Improved (after repair)	CureFuzz Re-eval of AUT-Improved
CartPole	495.7 $\pm$ 6.8	500 $\pm$ 0.0	77.4 $\pm$ 10.3	0.0(- 100.0%) $\pm$ 0.0	0.0 $\pm$ 0.0
MountainCar	-129.8 $\pm$ 2.8	-125.4 $\pm$ 3.2	167.2 $\pm$ 8.7	0.0(- 100.0%) $\pm$ 0.0	37.4 $\pm$ 6.4
Acrobot	-81.3 $\pm$ 3.0	-75.2 $\pm$ 2.9	1183.6 $\pm$ 41.9	118.2(- 90.0%) $\pm$ 11.3	265.0 $\pm$ 18.5
Hopper	3162.3 $\pm$ 60.9	3247.0 $\pm$ 30.5	154.4 $\pm$ 9.9	40.0(- 74.1%) $\pm$ 6.0	106.2 $\pm$ 19.5
Walker2D	3925.5 $\pm$ 192.2	4399.3 $\pm$ 93.8	251.8 $\pm$ 23.8	5.6(- 97.8%) $\pm$ 2.4	105.0 $\pm$ 17.2

Q-Learning (CQL) for this purpose. By applying the CQL objective, we can update the agent's existing policy using only the static data, ensuring the fine-tuning process is both stable and robust. Finally, we refer to this fine-tuned agent as *AUT-Improved*.

Result. We evaluate AUT-Improved in the same environment used to train the original AUT, with no modifications to the transition dynamics, reward function, or action space. The primary metric for comparison is the mean cumulative reward, averaged over 100 independent runs. The experiment results are detailed in Table 5, where AUT-Improved shows a consistent performance increase against the original AUT across all five environments. The enhancements were particularly notable in several cases: the CartPole agent achieved the theoretical maximum score of 500 with perfect stability (± 0.0), and the Walker2D agent's performance increased by over 12%. These results confirm that the optimality bugs identified by DELTA are highly useful for improving cumulative reward.

Repairing the Safety Failure. We further experiment with the robustness of AUT-Improved. For each environment, AUT-Improved is evaluated again on the known safety failure-inducing scenarios from the safety testing stage (the safety testing stage from RQ1). The result demonstrates the enhanced robustness of the AUT. AUT-Improved exhibits a substantial reduction in the number of such failures. In CartPole and MountainCar, all safety failures are repaired. Furthermore, AUT-Improved in Walker2D reduces its safety failures by more than 97%. These results demonstrate that our fine-tuning process not only leads to policies that are higher-performing but also significantly more robust.

Re-evaluation with CureFuzz. To further validate the robustness of the AUT-Improved, we launch a new fuzzing campaign with CureFuzz against the AUT-Improved. The fuzzing campaign follows the same execution setup described in RQ1. Table 5 presents the results. AUT-Improved consistently exhibits fewer safety failures than the original AUT. Notably, in CartPole, the AUT-Improved is completely robust, with zero safety failures detected. In Acrobot, the number of failures is reduced by approximately 77.6% (from 1183.6 to 265).

Answer to RQ4: The identified optimality bugs are highly useful in improving the performance of the AUT. An independent re-evaluation with CureFuzz confirms that the AUT-Improved is substantially more robust, reducing safety failures by up to 100% across all environments.

Table 6. Number of behavioral patterns detected by DELTA across the five environments at three relative distance thresholds $\tau \in \{0.10, 0.20, 0.30\}$. Results are averaged over five independent runs; $\pm$ values denote the 95% confidence interval margin of error.

τ	CartPole	MountainCar	Acrobot	Hopper	Walker2D
0.10	100.2 $\pm$ 15.2	4.2 $\pm$ 1.0	279.0 $\pm$ 10.5	1029.8 $\pm$ 139.0	1505.6 $\pm$ 204.6
0.20	25.4 $\pm$ 3.4	3.2 $\pm$ 0.6	130.6 $\pm$ 8.9	93.8 $\pm$ 6.4	1111.2 $\pm$ 152.7
0.30	3.8 $\pm$ 1.4	3.0 $\pm$ 0.0	54.8 $\pm$ 2.4	56.6 $\pm$ 5.9	445.2 $\pm$ 116.3

6 Discussion

6.1 Behavior Clustering

In this subsection, we analyze the AUT's behavior on bug-triggering initial states. We group the AUT's trajectories from these initial states into clusters and treat each cluster as a behavioral pattern, following prior work [37]. Specifically, we adopt unsupervised hierarchical clustering [50]: Each trajectory initially forms a singleton cluster, and at every step, the pair of clusters with the smallest pairwise distance is merged. This bottom-up process continues until all trajectories collapse into a single cluster, yielding a dendrogram with maximum merge distance $d_{\max}$. For pairwise distance, we use Dynamic Time Warping (DTW) [3] computed over the state sequences, which natively accommodates trajectories of varying length without padding. Unlike K-means [40], which requires the number of clusters to be fixed in advance, hierarchical clustering imposes no target count; clusters emerge naturally from the trajectory structure given a cut threshold τ , expressed as a fraction of $d_{\max}$. We report results at three values, $\tau \in \{0.10, 0.20, 0.30\}$, in Table 6.

Findings. The number of behavioral patterns varies with both the environment and the cut threshold τ , decreasing monotonically as τ grows. MountainCar yields only 3–4 behavioral patterns across the three thresholds, reflecting the environment's low-dimensional state space and small discrete action space. In contrast, higher-dimensional environments such as Walker2D exhibit hundreds of distinct behavioral patterns even at $\tau = 0.30$.

6.2 Extension to Non-Deterministic Environments

Our experiment setting assumes a deterministic environment, which is the same setting as in previous work [14, 85]. However, many real-world environments exhibit stochasticity due to random disturbances or noisy dynamics. To demonstrate that DELTA remains applicable in such settings, we present a natural extension of our oracle formulation based on statistical hypothesis testing, and empirically validate it on a non-deterministic benchmark.

Statistical Oracle Formulation. For a given initial state s_0 , we execute both the AUT and the challenger agent N times (default $N = 5$), obtaining two sets of cumulative rewards: $\mathcal{R}_{\text{aut}} = \{R_{\text{aut}}^{(1)}, \dots, R_{\text{aut}}^{(N)}\}$ and $\mathcal{R}_{\text{ca}} = \{R_{\text{ca}}^{(1)}, \dots, R_{\text{ca}}^{(N)}\}$. An optimality bug is reported when the challenger's rewards are statistically significantly higher than the AUT's:

$$\text{OptimalityBug}(s_0) = \# [p < \alpha] \quad (9)$$

where p is the p-value from a one-sided Mann-Whitney U test [48] with the alternative hypothesis $H_1 : R_{\text{ca}} > R_{\text{aut}}$, and α is the significance level (default $\alpha = 0.05$).

Empirical Validation. To demonstrate the effectiveness of the statistical oracle, we conducted additional experiments on *Hopper-Noise*, a non-deterministic variant of Hopper that introduces random wind disturbances [42, 57]. Following the same protocol as RQ1, the AUT achieves a mean reward of 3086.9 ± 61.0 over 100 episodes, with 477.4 ± 91.2 safety failures detected during safety testing. For optimality testing, we adopt the same two-hour budget per run and repeat

Table 7. Optimality bug detection on the non-deterministic Hopper-Noise environment. Top: comparison of offline RL algorithms for training the challenger. Bottom: DELTA against baseline methods. All results are averaged over five runs; $\pm$ values denote the 95% confidence interval margin of error.

Challenger Algorithm	BC	BCQ	CQL	
# Bugs	44.6 $\pm$ 11.3	87.8 $\pm$ 23.3	65.2 $\pm$ 22.8	
Testing Method	DELTA	CureFuzz	GMT	MDPFuzz
# Bugs	87.8 $\pm$ 23.3 (+171%)	32.4 $\pm$ 11.3	25.2 $\pm$ 6.7	24.6 $\pm$ 6.4

each experiment five times, with $N = 5$ rollouts per initial state for the statistical oracle. Table 7 reports the number of optimality bugs detected. BCQ remains the strongest offline RL algorithm for training the challenger. Using the BCQ challenger, DELTA detects 171% more optimality bugs than the best-performing baseline, confirming that our differential testing methodology remains effective and statistically sound under non-determinism.

6.3 Reliance on Thorough Safety Testing

The challenger’s quality depends on the diversity of the safety-testing dataset $\mathcal{D}$ used to train it. As long as safety testing is conducted thoroughly, $\mathcal{D}$ can be trusted to be diverse; if not, this would imply that safety testing itself has failed to adequately explore the AUT’s behavior—a far more serious problem than missed optimality issues. DELTA therefore operates under an implicit precondition: it should be applied only when developers are satisfied that safety testing has been rigorous enough to expose the AUT’s safety-critical failures. Under this precondition, $\mathcal{D}$ can be reasonably regarded as diverse, and DELTA’s optimality testing serves as a complementary layer on top of safety testing. In practice, we adopt a two-hour safety-testing budget per environment, consistent with prior work [23, 44, 83]. Developers working with more complex environments may need to extend this budget accordingly. How long a testing campaign should run is itself a long-standing open question in the software testing community [4, 39].

6.4 Practical Considerations

When Can the Challenger Surpass the AUT? The challenger can outperform the AUT when: (1) the AUT has not converged to the optimal policy, and room exists for an alternative policy to outperform it in certain regions of the state space; (2) the safety testing dataset naturally contains diverse scenarios including edge cases. This diversity mitigates distributional shift and enables *stitching* of suboptimal trajectory segments into improved policies [19]. Overall, the challenger need not outperform the AUT globally. It functions as a *local* oracle: surpassing the AUT from specific initial states suffices to reveal faults.

When the Challenger Fails to Outperform. Importantly, this oracle yields value regardless of outcome. A superior challenger exposes optimality issues and guides policy improvement. When the AUT is already near-optimal, the challenger is unlikely to outperform it. This does not indicate oracle failure; rather, it provides empirical confirmation of the high performance of the AUT. The goal of testing is to assess the optimality, not necessarily to find faults. Confirming that the AUT performs well is as valuable as uncovering defects.

Training Efficiency. Our efficiency comparison (Section 5) reports only offline training time, excluding the data collection performed during safety testing. Safety testing is a prerequisite for deploying DRL systems, and is not optional, as safety concerns take precedence over optimality. The data collection period is therefore a zero-cost byproduct of an already-required process.

Repair vs. Replace. RQ4 shows that the optimality bugs detected by DELTA have practical value for repairing the AUT via fine-tuning. A natural question is whether the challenger should instead be deployed directly as the final agent. We view fine-tuning and replacement as complementary downstream choices enabled by DELTA’s bug detection. When the AUT’s weights are inaccessible, fine-tuning is not feasible, and the challenger becomes a candidate for direct deployment. The choice between these modes is ultimately a deployment decision determined by access constraints and operational requirements.

7 Threats to Validity

One threat to *internal validity* is the randomness of the testing algorithms used in our experiments. To mitigate it, we repeat each experiment five times and use appropriate statistical tests to account for both statistical significance and effect size. When implementing CureFuzz [26], MDPFuzz [55], and GMT [38], we reuse their officially released replication packages. In terms of *external validity*, one threat is that the results of our analysis may not be generalizable. To mitigate it, we have carefully selected diverse tasks and diverse AUTs to evaluate our method, which ensures that our results are not biased. One potential threat to *construct validity* is that the evaluation metrics may not fully capture the performance of our baselines. The number of bugs, distance, and distinct bugs are widely adopted metrics for fuzzing DRL agents [26, 45, 55].

A further threat to construct validity concerns how optimality bugs are counted. Following established practice in DRL testing [26, 45, 55], we count bugs at the granularity of the initial state, as initial states are concrete, reproducible inputs that developers can directly use for debugging and retraining. However, counting at this granularity does not directly reflect the diversity of underlying agent behaviors. To complement this initial-state count, we additionally report a behavioral-pattern count in Section 6.1. Another threat to construct validity concerns the possibility of false negatives. DELTA’s challenger is trained on data collected during safety testing; if the safety tester underexplores certain regions of the state space, the challenger may be undertrained in those regions and fail to surpass the AUT even where the AUT is genuinely suboptimal. To mitigate this, DELTA is designed to integrate with any state-of-the-art safety testing method, and our current implementation adopts CureFuzz [26], which explicitly promotes exploration diversity via a curiosity-driven mechanism. We further discuss this dependency in Section 6.3. Nonetheless, DELTA’s reported bug counts should be interpreted as a sound-but-not-complete lower bound on the AUT’s optimality issues.

8 Related Work

Safety Testing of DRL Agents. Pang et al. introduce MDPFuzz [55], which represents the pioneering effort in black-box fuzz testing for deep learning models addressing Markov Decision Processes, of which DRL agents are a straightforward application. He et al. further propose CureFuzz [26]. CureFuzz proposes a curiosity mechanism to measure the novelty of a scenario, which aims to reveal a diverse set of crash-triggering scenarios. Li et al. [38] present GMT, a testing framework that utilizes a generative diffusion model [80] to generate practical test cases and novelty-based guidance, thereby diversifying agent behaviors and improving test effectiveness. Shi et al. [62] propose Synthify, which accelerates the falsification of AI-enabled control systems by synthesizing lightweight proxy programs that approximate the neural controller. Complementary to offline testing, they further synthesize efficient and permissive programmatic runtime shields that correct unsafe actions of neural policies at deployment time [61]. As multi-agent systems are increasingly adopted for complex real-world tasks [25], testing their reliability has attracted growing attention. Ma et al. [44] focus on testing multi-agent DRL systems and propose MASTest, which incorporates

both individual diversity and team diversity. Furthermore, they propose AdvTest [45] to evaluate competitive game agents through constraint-guided adversarial agent training.

Search-Based Testing of DRL Agents. Zolfagharian et al. [85] propose STARLA, which uses a genetic algorithm to find faulty episodes and actions leading to crashes. Tappler et al. [72] introduce a search-based testing approach for RL agents with stochastic policies. Their method employs a depth-first backtracking search algorithm to identify reference traces that solve RL tasks and boundary states leading to unsafe states. Tappler et al. [73] further present a framework that integrates search-based testing with Reinforcement Learning from Demonstrations (RLfD) [51].

Mutation Testing of DRL Agents. Tambon et al. [69] and Lu et al. [41] introduce several RL-specific mutation operators. Thomas et al. [74] introduce muPRL, which utilizes a systematic taxonomy of real-world faults to design specialized mutation operators and evaluate the effectiveness of testing suites in detecting realistic DRL bugs.

Metamorphic Testing of DRL Agents. Eniser et al. apply metamorphic testing using manually designed relaxations as metamorphic oracles to assess DRL policies [14]. Subsequently, Eisenhut et al. introduce new metamorphic oracles alongside a search-based testing method to identify bugs in DRL agents [13]. More recently, Cheng et al. propose Decictor [6], a method designed to generate non-optimal decision scenarios, specifically in cases where an autonomous driving system fails to plan optimal paths for autonomous vehicles (AVs). A limitation of Decictor is its reliance on pre-existing knowledge of some optimal paths and domain-specific metamorphic relations. In contrast, our work tackles optimality issues using a differential testing approach, making it more broadly applicable.

9 Conclusion and Future Work

In this paper, we introduced DELTA, a two-phase testing framework that automatically identifies both safety-critical and optimality bugs. DELTA first conducts safety testing on the AUT while collecting its interaction trajectories. Subsequently, these trajectories are used to train a challenger agent via Offline Reinforcement Learning; if the challenger achieves higher cumulative rewards, an optimality issue is flagged in the AUT. Our extensive evaluations across five diverse environments demonstrated DELTA's effectiveness in uncovering optimality issues. Challenger agents trained with BCQ proved most effective for DELTA in identifying an average of 2,518 optimality issues per environment, and DELTA significantly outperformed baseline methods. In future work, we plan to evaluate DELTA on more complex real-world domains, such as autonomous driving, and to explore emerging offline RL algorithms for training stronger challenger agents.

10 Data Availability

A replication package, including source code, experimental datasets, and detailed results, is available on Figshare [2] and can be accessed at <https://doi.org/10.6084/m9.figshare.29196884.v1>.

References

- [1] 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. doi:10.1109/msp.2017.2743240
- [2] Anonymous Authors. 2026. Replication Package for "Learning from the Test: Self-Referential Differential Testing for Deep RL Agents". doi:10.6084/m9.figshare.29196884.v1
- [3] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. 359–370.
- [4] Marcel Böhme. 2019. Assurances in software testing: A roadmap. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 5–8. doi:10.1109/icse-nier.2019.00010

- [5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. Exploration by random network distillation. In *International Conference on Learning Representations*. doi:10.48550/arXiv.1810.12894
- [6] Mingfei Cheng, Xiaofei Xie, Yuan Zhou, Junjie Wang, Guozhu Meng, and Kairui Yang. 2025. *Decictor: Towards Evaluating the Robustness of Decision-Making in Autonomous Driving Systems*. IEEE Press, 424–436. doi:10.1109/icse55347.2025.00114
- [7] Jacob Cohen. 1992. Statistical power analysis. *Current Directions in Psychological Science* 1, 3 (1992), 98–101. doi:10.1111/1467-8721.ep10768783
- [8] Antonio Coronato, Muddasar Naeem, Giuseppe De Pietro, and Giovanni Paragliola. 2020. Reinforcement learning for intelligent healthcare applications: A survey. *Artificial Intelligence in Medicine* 109 (2020), 101964. doi:10.1016/j.artmed.2020.101964
- [9] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2, 4 (1989), 303–314. doi:10.1007/BF02551274
- [10] Per-Erik Danielsson. 1980. Euclidean distance mapping. *Computer Graphics and Image Processing* 14, 3 (1980), 227–248. doi:10.1016/0146-664x(80)90054-4
- [11] Zihan Ding and Hao Dong. 2020. Challenges of reinforcement learning. *Deep Reinforcement Learning: Fundamentals, Research and Applications* (2020), 249–272. doi:10.1007/978-981-15-4095-0_7
- [12] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning* 110, 9 (2021), 2419–2468. doi:10.1007/s10994-021-05961-4
- [13] Jan Eisenhut, Álvaro Torralba, Maria Christakis, and Jörg Hoffmann. 2023. Automatic metamorphic test oracles for action-policy testing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 33. 109–117. doi:10.1609/icaps.v33i1.27185
- [14] Hasan Ferit Eniser, Timo P Gros, Valentin Wüstholtz, 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. doi:10.1145/3533767.3534392
- [15] Robert B Evans and Alberto Savoia. 2007. Differential testing: a new approach to change detection. In *The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*. 549–552. doi:10.1145/1295014.1295038
- [16] Xing Fang, Qichao Zhang, Yinfeng Gao, and Dongbin Zhao. 2022. Offline reinforcement learning for autonomous driving with real-world driving data. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3417–3422. doi:10.1109/itsc55140.2022.9922100
- [17] Farama Foundation. 2024. Acrobot. https://gymnasium.farama.org/environments/classic_control/acrobot/.
- [18] Farama Foundation. 2024. Hopper. <https://gymnasium.farama.org/environments/mujoco/hopper/>.
- [19] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219* (2020). doi:10.48550/arXiv.2004.07219
- [20] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürri. 2021. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters* 6, 3 (2021), 4257–4264. doi:10.1109/lra.2021.3064284
- [21] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*. PMLR, 2052–2062. doi:10.48550/arXiv.1812.02900
- [22] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*. PMLR, 1861–1870. doi:10.48550/arXiv.1801.01290
- [23] Fitash Ul Haq, Donghwan Shin, and Lionel C Briand. 2023. Many-objective reinforcement learning for online testing of dnn-enabled systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1814–1826. doi:10.1109/icse48619.2023.00155
- [24] Junda He, Jieke Shi, Terry Yue Zhuo, Christoph Treude, Jiamou Sun, Zhenchang Xing, Xiaoning Du, and David Lo. 2026. LLM-as-a-judge for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* (2026). doi:10.1145/3797276
- [25] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30. doi:10.1145/3712003
- [26] 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 *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–14. doi:10.1145/3597503.3639149
- [27] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. 2021. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research* 40, 4–5

- (2021), 698–721. doi:10.1177/0278364920987859
- [28] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. *arXiv preprint arXiv:2102.08942* (2021). doi:10.48550/arXiv.2102.08942
- [29] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems* 23, 6 (2021), 4909–4926. doi:10.1109/TITS.2021.3054625
- [30] Haruka Kiyohara, Ren Kishimoto, Kosuke Kawakami, Ken Kobayashi, Kazuhide Nakata, and Yuta Saito. 2024. Towards assessing and benchmarking risk-return tradeoff of off-policy evaluation. In *International Conference on Learning Representations*. doi:10.48550/arXiv.2311.18207
- [31] 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. doi:10.1177/0278364913495721
- [32] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. 2022. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*. doi:10.48550/arXiv.2110.06169
- [33] Aviral Kumar, Anikait Singh, Frederik Ebert, Mitsuhiko Nakamoto, Yanlai Yang, Chelsea Finn, and Sergey Levine. 2023. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. In *Robotics: Science and Systems*. doi:10.15607/RSS.2023.XIX.019
- [34] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191. doi:10.48550/arXiv.2006.04779
- [35] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. 2022. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*. PMLR, 1702–1712. doi:10.48550/arXiv.2107.00591
- [36] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020). doi:10.48550/arXiv.2005.01643
- [37] Zechu Li, Rickmer Krohn, Tao Chen, Anurag Ajay, Pulkit Agrawal, and Georgia Chalvatzaki. 2024. Learning multimodal behaviors from scratch with diffusion policy gradient. *Advances in Neural Information Processing Systems* 37 (2024), 38456–38479. doi:10.52202/079017-1214
- [38] Zhuo Li, 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. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 243–254. doi:10.1109/ase56229.2023.00153
- [39] Stephan Lipp, Daniel Elsner, Severin Kacianka, Alexander Pretschner, Marcel Böhme, and Sebastian Banescu. 2023. Green fuzzing: A saturation-based stopping criterion using vulnerability prediction. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 127–139. doi:10.1145/3597926.3598043
- [40] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137. doi:10.1109/tit.1982.1056489
- [41] Yuteng Lu, Weidi Sun, and Meng Sun. 2022. Towards mutation testing of reinforcement learning systems. *Journal of Systems Architecture* 131 (2022), 102701. doi:10.1016/j.sysarc.2022.102701
- [42] Baiting Luo, Ava Pettet, Aron Laszka, Abhishek Dubey, and Ayan Mukhopadhyay. 2025. Scalable Decision-Making in Stochastic Environments through Learned Temporal Abstraction. In *The Thirteenth International Conference on Learning Representations*. doi:10.48550/arXiv.2502.21186
- [43] 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. doi:10.1145/3238147.3238202
- [44] Xuyan Ma, Yawen Wang, Junjie Wang, Xiaofei Xie, Boyu Wu, Shoubin Li, Fanjiang Xu, and Qing Wang. 2024. Enhancing Multi-agent System Testing with Diversity-Guided Exploration and Adaptive Critical State Exploitation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1491–1503. doi:10.1145/3650212.3680376
- [45] Xuyan Ma, Yawen Wang, Junjie Wang, Xiaofei Xie, Boyu Wu, Yiguang Yan, Shoubin Li, Fanjiang Xu, and Qing Wang. 2024. Diversity-Oriented Testing for Competitive Game Agent via Constraint-Guided Adversarial Agent Training. *IEEE Transactions on Software Engineering* (2024). doi:10.1109/tse.2024.3491193
- [46] Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. 2022. Deep learning, reinforcement learning, and world models. *Neural Networks* 152 (2022), 267–275. doi:10.1016/j.neunet.2022.03.037
- [47] William M McKeeman. 1998. Differential testing for software. *Digital Technical Journal* 10, 1 (1998), 100–107.
- [48] Patrick E McKnight and Julius Najab. 2010. Mann-Whitney U Test. *The Corsini Encyclopedia of Psychology* (2010), 1–1. doi:10.1002/9780470479216.corpsy0524

- [49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:[1312.5602](https://arxiv.org/abs/1312.5602) doi:[10.48550/arXiv.1312.5602](https://doi.org/10.48550/arXiv.1312.5602)
- [50] Fionn Murtagh and Pedro Contreras. 2012. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 1 (2012), 86–97. doi:[10.1002/widm.53](https://doi.org/10.1002/widm.53)
- [51] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2018. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 6292–6299. doi:[10.1109/icra.2018.8463162](https://doi.org/10.1109/icra.2018.8463162)
- [52] Hai Nguyen and Hung La. 2019. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 590–595. doi:[10.1109/irc.2019.00120](https://doi.org/10.1109/irc.2019.00120)
- [53] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. 2020. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics* 50, 9 (2020), 3826–3839. doi:[10.1109/tcyb.2020.2977374](https://doi.org/10.1109/tcyb.2020.2977374)
- [54] Tom Le Paine, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, et al. 2020. Making efficient use of demonstrations to solve hard exploration problems. In *International Conference on Learning Representations*. doi:[10.48550/arXiv.1909.01387](https://doi.org/10.48550/arXiv.1909.01387)
- [55] Qi Pang, Yuanyuan Yuan, and Shuai Wang. 2022. Mdpfuzz: testing models solving markov decision processes. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 378–390. doi:[10.1145/3533767.3534388](https://doi.org/10.1145/3533767.3534388)
- [56] Martin L Puterman. 1990. Markov decision processes. *Handbooks in Operations Research and Management Science* 2 (1990), 331–434. doi:[10.1016/S0927-0507\(05\)80172-0](https://doi.org/10.1016/S0927-0507(05)80172-0)
- [57] Marc Rigter, Bruno Lacerda, and Nick Hawes. 2023. One risk to rule them all: A risk-sensitive perspective on model-based offline reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 77520–77545. doi:[10.52202/075280-3389](https://doi.org/10.52202/075280-3389)
- [58] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. In *IS&T International Symposium on Electronic Imaging: Autonomous Vehicles and Machines*. 70–76. doi:[10.2352/ISSN.2470-1173.2017.19.AVM-023](https://doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023)
- [59] Takuma Seno and Michita Imai. 2022. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research* 23, 315 (2022), 1–20. doi:[10.48550/arXiv.2111.03788](https://doi.org/10.48550/arXiv.2111.03788)
- [60] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. 2016. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295* (2016). doi:[10.48550/arXiv.1610.03295](https://doi.org/10.48550/arXiv.1610.03295)
- [61] Jieke Shi, Junda He, Zhou Yang, Đorđe Žikelić, and David Lo. 2025. Synthesizing Efficient and Permissive Programmatic Runtime Shields for Neural Policies. *ACM Transactions on Software Engineering and Methodology* (2025). doi:[10.1145/3773034](https://doi.org/10.1145/3773034)
- [62] Jieke Shi, Zhou Yang, Junda He, Bowen Xu, Dongsun Kim, DongGyun Han, and David Lo. 2025. Finding Safety Violations of AI-Enabled Control Systems through the Lens of Synthesized Proxy Programs. *ACM Transactions on Software Engineering and Methodology* (2025). doi:[10.1145/3715105](https://doi.org/10.1145/3715105)
- [63] Tianyu Shi, Dong Chen, Kaian Chen, and Zhaojian Li. 2021. Offline reinforcement learning for autonomous driving with safety and exploration enhancement. *arXiv preprint arXiv:2110.07067* (2021). doi:[10.48550/arXiv.2110.07067](https://doi.org/10.48550/arXiv.2110.07067)
- [64] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. 2022. S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*. PMLR, 907–917. doi:[10.48550/arXiv.2103.06326](https://doi.org/10.48550/arXiv.2103.06326)
- [65] Aaron Sonabend-W, Junwei Lu, Leo Anthony Celi, Tianxi Cai, and Peter Szolovits. 2020. Expert-supervised reinforcement learning for offline policy learning and evaluation. *Advances in Neural Information Processing Systems* 33 (2020), 18967–18977. doi:[10.48550/arXiv.2006.13189](https://doi.org/10.48550/arXiv.2006.13189)
- [66] Wesley A Suttle, Aamodh Suresh, and Carlos Nieto-Granda. 2025. Behavioral Entropy-Guided Dataset Generation for Offline Reinforcement Learning. *arXiv preprint arXiv:2502.04141* (2025). doi:[10.48550/arXiv.2502.04141](https://doi.org/10.48550/arXiv.2502.04141)
- [67] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
- [68] Victor Talpaert, Ibrahim Sobh, B Ravi Kiran, Patrick Mannion, Senthil Yogamani, Ahmad El-Sallab, and Patrick Perez. 2019. Exploring applications of deep reinforcement learning for real-world autonomous driving systems. In *International Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*. 564–572. doi:[10.48550/arXiv.1901.01536](https://doi.org/10.48550/arXiv.1901.01536)
- [69] Florian Tambon, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, and Giuliano Antoniol. 2023. Mutation testing of deep reinforcement learning based on real faults. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 188–198. doi:[10.1109/icst57152.2023.00026](https://doi.org/10.1109/icst57152.2023.00026)
- [70] Shengpu Tang, Maggie Makar, Michael Sjoding, Finale Doshi-Velez, and Jenna Wiens. 2022. Leveraging factored action spaces for efficient offline reinforcement learning in healthcare. *Advances in Neural Information Processing Systems* 35 (2022), 34272–34286. doi:[10.52202/068431-2484](https://doi.org/10.52202/068431-2484)

- [71] Shengpu Tang and Jenna Wiens. 2021. Model selection for offline reinforcement learning: Practical considerations for healthcare settings. In *Machine Learning for Healthcare Conference*. PMLR, 2–35. doi:10.48550/arXiv.2107.11003
- [72] 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. doi:10.24963/ijcai.2022/72
- [73] Martin Tappler, Andrea Pferscher, Bernhard K Aichernig, and Bettina Könighofer. 2024. Learning and Repair of Deep Reinforcement Learning Policies from Fuzz-Testing Data. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13. doi:10.1145/3597503.3623311
- [74] Deepak-George Thomas, Matteo Biagiola, Nargiz Humbatova, Mohammad Wardat, Gunel Jahangirova, Hridesh Rajan, and Paolo Tonella. 2025. muPRL: A Mutation Testing Pipeline for Deep Reinforcement Learning based on Real Faults. In *IEEE/ACM International Conference on Software Engineering*. doi:10.1109/ICSE55347.2025.00036
- [75] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral cloning from observation. In *International Joint Conference on Artificial Intelligence*. 4950–4957. doi:10.48550/arXiv.1805.01954
- [76] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 2094–2100. doi:10.1609/AAAI.V30I1.10295
- [77] Dennis D Wackerly. 2008. *Mathematical statistics with applications*. Thomson Brooks/Cole.
- [78] Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. 2021. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 27395–27407. doi:10.48550/arXiv.2106.04895
- [79] Ahmed Haj Yahmed, Rached Bouchoucha, Housseem Ben Braiek, and Foutse Khomh. 2023. An Intentional Forgetting-Driven Self-Healing Method For Deep Reinforcement Learning Systems. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1314–1325. doi:10.1109/ase56229.2023.00121
- [80] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. 2023. Diffusion models: A comprehensive survey of methods and applications. *Comput. Surveys* 56, 4 (2023), 1–39. doi:10.1145/3626235
- [81] 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*. 1482–1493. doi:10.1145/3510003.3510146
- [82] Chao Yu, Jiming Liu, Shamim Nemat, and Guosheng Yin. 2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–36. doi:10.1145/3477600
- [83] Yan Zheng, Xiaofei Xie, Ting Su, Lei Ma, Jianye Hao, Zhaopeng Meng, Yang Liu, Ruimin Shen, Yingfeng Chen, and Changjie Fan. 2019. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 772–784. doi:10.1109/ase.2019.00077
- [84] Xin Zhou, Bowen Xu, Kisub Kim, DongGyun Han, Hung Huu Nguyen, Thanh Le-Cong, Junda He, Bach Le, and David Lo. 2024. Leveraging Large Language Model for Automatic Patch Correctness Assessment. *IEEE Transactions on Software Engineering* 50, 11 (2024), 2865–2883. doi:10.1109/tse.2024.3452252
- [85] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C Briand, Mojtaba Bagherzadeh, and S Ramesh. 2023. A search-based testing approach for deep reinforcement learning agents. *IEEE Transactions on Software Engineering* (2023). doi:10.1109/tse.2023.3269804

Received 2026-01-30; accepted 2026-06-25