MDPMorph: An MDP-Based Metamorphic Testing Framework for Deep Reinforcement Learning Agents

Jiapeng Li*, Zheng Zheng*, Yuning Xing†, Daixu Ren*, Steven Cho†, and Valerio Terragni†
*Beihang University, Beijing, China
Email: {jp_li, zhengz, rendaixu}@buaa.edu.cn
†University of Auckland, Auckland, New Zealand
Email: {yxin683, scho518, vter674}@aucklanduni.ac.nz

***Abstract*—Deep Reinforcement Learning (DRL) systems are widely used across various domains. However, testing these systems presents significant challenges. The DRL agent, which serves as the core decision-maker, generates continuous value estimates rather than discrete labels and operates under non-stationary policies within complex and stochastic environments. Consequently, there is no definitive "correct answer" for each state-action pair, complicating automated test generation due to the known oracle problem. To address this challenge, we propose a Metamorphic Testing (MT) framework (MDPMorph) specifically designed for validating DRL agents. Our framework is based on Markov Decision Processes (MDP) and focuses on the core reasoning properties of agents to automatically uncover potential faults. To support MDPMorph, we introduce a Metamorphic Relation (MR) design methodology tailored for DRL agents, based on the temporal characteristics of MDP. Using this method, we define nine generic MRs that encapsulate common and expected properties of an agent's reasoning process. Furthermore, based on established assumptions and definitions within the context of MDPs, we theoretically demonstrate the soundness of these MRs. Finally, we specialize these generic MRs into environment-specific MRs by determining appropriate thresholds through training on three classic DRL environments. Our experimental results demonstrate that MDPMorph and the proposed MRs are highly effective in automatically detecting mutants within these studied environments, with a 0.84 average mutation detection rate.**

***Index Terms*—Deep Reinforcement Learning, Metamorphic Testing, Machine Learning Testing, SE4AI**

I. Introduction

Over the past decade, Reinforcement Learning (RL) has gained significant attention for its ability to solve challenging sequential decision-making problems [1]. By combining RL with Deep Neural Networks (DNNs), **Deep Reinforcement Learning (DRL)** harnesses the representational power of DNNs to tackle tasks of even greater complexity and uncertainty without extensive domain knowledge [2]. Deep architectures enable DRL agents to automatically extract multi-level feature hierarchies from raw inputs, driving breakthroughs across a wide range of applications [3]. DRL has been successfully deployed in fields such as autonomous driving [4], [5], robotic control [6], and intelligent transportation systems [7].

Similar to other software systems, the agent (acting as the central decision-making component in an DRL system) faces considerable challenges in ensuring correct decisions, particularly in complex and uncertain real-world environments. Any potential fault that causes the agent to make an incorrect decision could lead to unexpected, or even catastrophic consequences. In autonomous driving, for instance, the result could range from vehicle damage to human casualties [8]. Therefore, it is essential to thoroughly evaluate safety-critical DRL agents under a wide range of failure scenarios to identify and address hidden vulnerabilities before deploying them in real-world environments. Recent research has explored software testing techniques for DRL systems that automatically generate diverse test cases to identify as many faults in agents as possible [9], [10], [11], [12]. However, these testing methods rely on human-written oracles, the oracle problem remains a major challenge in automated testing of DRL systems [13].

Metamorphic Testing (MT) [14] is a popular software testing technique that, when combined with automated test generation, allows for the automatic detection of faults without requiring the expected output for each individual test case. This is achieved by checking whether the outputs of related inputs satisfy specific conditions known as **Metamorphic Relations (MRs)**. Significant research has been done on MT for traditional software [15], [16] and Machine Learning (ML) [17] systems. In the field of ML, especially, the high computational complexity and vast input space make it difficult for software developers to design systematic testing methods for thorough evaluation and quality assurance. As a result, MT has emerged as a promising testing technique and has been extensively applied to ML programs [18], [16]. However, existing MT methods and MRs are inadequate for DRL systems due to their inherent characteristics. These fall into three main categories: First, although MT has been applied to both traditional software and ML models, it often emphasizes single-step input–output relationships. In contrast, the behavior of an DRL agent unfolds over time, which makes multi-step or trajectory-level MRs particularly important. Second, unlike traditional software or ML models that typically operate in static and predictable environments, an DRL agent interacts with a dynamic and complex environment, making it challenging for previous MT methods to effectively describe and verify the agent's policy behavior. Third, agents within DRL systems generate data that is uncertain and interactive during the reasoning phase, such

Corresponding authors: Zheng Zheng (zhengz@buaa.edu.cn) and Valerio Terragni (v.terragni@auckland.ac.nz)

This is the authors' version of the paper that was accepted for publication in the research track of The 36th IEEE International Symposium on Software Reliability Engineering (ISSRE 2025)

as state sequences, action sequences, and reward sequences. Existing MT methods lack the ability to recognize and process these data streams, as they were not designed for this purpose.

To tackle these challenges, we propose **MDPMORPH**, a novel MT framework for DRL systems, to effectively alleviate the oracle problem in DRL agents. Grounded in the **Markov Decision Process (MDP)**, our framework considers the internal logical structure of the environment while incorporating the agent's decision-making behavior. Although this framework could potentially be applied to a broader range of RL agents, the increased nonlinearity, deep network architectures, and large-scale parameters of DRL agents exacerbate the oracle problem. As a result, the MDPMORPH proposed in this paper is primarily designed for DRL agents. Since the MDP provides a rigorous mathematical foundation and verifiable temporal relationships for DRL systems, our MR design is primarily based on MDP properties. Specifically, we leverage the temporal characteristics of the MDP to divide the agent's inference process into local MDP and global MDP, where the local MDP is further categorized into single-step and multi-step. Based on these hierarchical levels, as well as considering MDP assumptions and properties, we propose nine MRs specifically tailored for DRL systems. We also provide theoretical verification to show that these relations represent essential properties of all DRL systems that meet the specified assumptions. Subsequently, following prior work on automatically generating MRs [19], [20], [21], we propose an automated threshold training method aiming for zero False Positives (FPs) and minimal False Negatives (FNs) to determine the optimal thresholds for the MRs under different environments.

We evaluated MDPMORPH and our proposed MRs across three widely used DRL environments (CARTPOLE, LUNARLANDER, and BIPEDALWALKER) using 1,000 metamorphic test pairs and 80 different mutants. Experimental results show that, although individual MRs may miss certain mutants, they collectively detects all mutants. The average mutation detection rate across the three environments is 0.73, 0.93, and 0.87, respectively. These results validate the effectiveness of MDPMORPH and MRs. Further analysis reveals that neuron-level and layer-level mutants are more difficult to detect, which explains why some MRs fail to detect them.

In summary, this paper makes the following contributions:

- A new MR design methodology for DRL systems that leverages the characteristics of MDPs. Using this methodology, we propose nine generic MRs for DRL agents and provide theoretical analysis of their necessary properties.
- MDPMORPH, a new MT framework for DRL agents that automatically learns thresholds within MRs, enabling the fully automated transformation of generic MRs into environment-specific ones.
- Experiments conducted on three classic DRL environments that demonstrate that our MRs exhibit strong mutant detection capabilities.
- We publicly release the tool implementation [22] and experimental data [23] to foster future research.

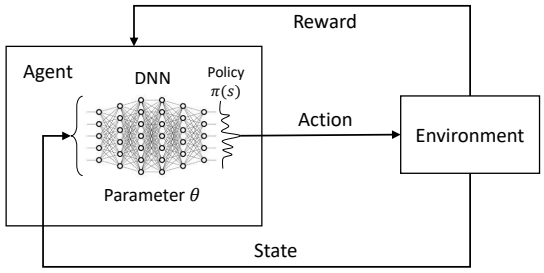

Figure 1. Structural diagram of Deep Reinforcement Learning (DRL)

II. PRELIMINARIES

A. Deep Reinforcement Learning (DRL) Systems

DRL systems are decision-making methods that incorporate RL with DNNs. By leveraging neural networks to approximate policy or value functions, these systems enable agents to learn optimal or near-optimal strategies in complex environments characterized by high dimensionality and continuity [24], [3]. Figure 1 illustrates the structural diagram of DRL systems.

The operation of an DRL system can be described as a **Markov Decision Process (MDP)** [25], a discrete-time stochastic control process used to model sequential decision-making problems. An MDP is defined by a tuple consisting of states s, actions a, state transition function $P(s, a)$, rewards r, and a discount factor. The system is composed of two principal entities: the environment and the agent. The environment, governed by its own dynamic rules, transitions the current state s to a new state s' upon receiving an action a from the agent, and provides scalar reward r through a reward function. The agent observes the state s of the environment at each time step and samples an action a according to its policy $\pi(s)$, continuously updating its network parameters based on the received rewards. The goal of training is to learn a policy that maximizes discounted cumulative reward, which is then used during the reasoning phase.

B. Metamorphic Testing (MT)

Metamorphic Testing (MT) [14] is an approach designed to alleviate the oracle problem. At the heart of MT are Metamorphic Relations (MRs), which define expected relationships among the outputs corresponding to multiple related inputs. The fundamental insight of MT is that, even when it is not feasible to automatically determine the correctness of a single output, *relationships among expected outputs from related inputs can be leveraged as test oracles* [16].

Definition 1: Let f denote the system under test. A **Metamorphic Relation (MR)** is a property involving multiple inputs $\langle x_1, \ldots, x_n \rangle$ and their respective outputs $\langle f(x_1), \ldots, f(x_n) \rangle$, where $n \geq 2$. Formally, an MR is a logical implication ($\Rightarrow$) [1]:

$$\mathcal{R}_i(x_1, \ldots, x_n) \Rightarrow \mathcal{R}_o(f(x_1), \ldots, f(x_n))$$

where $\mathcal{R}_i$ is the input and $\mathcal{R}_o$ is the output relation.

[1]A broader definition allows inputs and outputs to appear in both input and output relations [16]. However, without loss of generality, we use the conventional form as our proposed MR aligns with this definition.

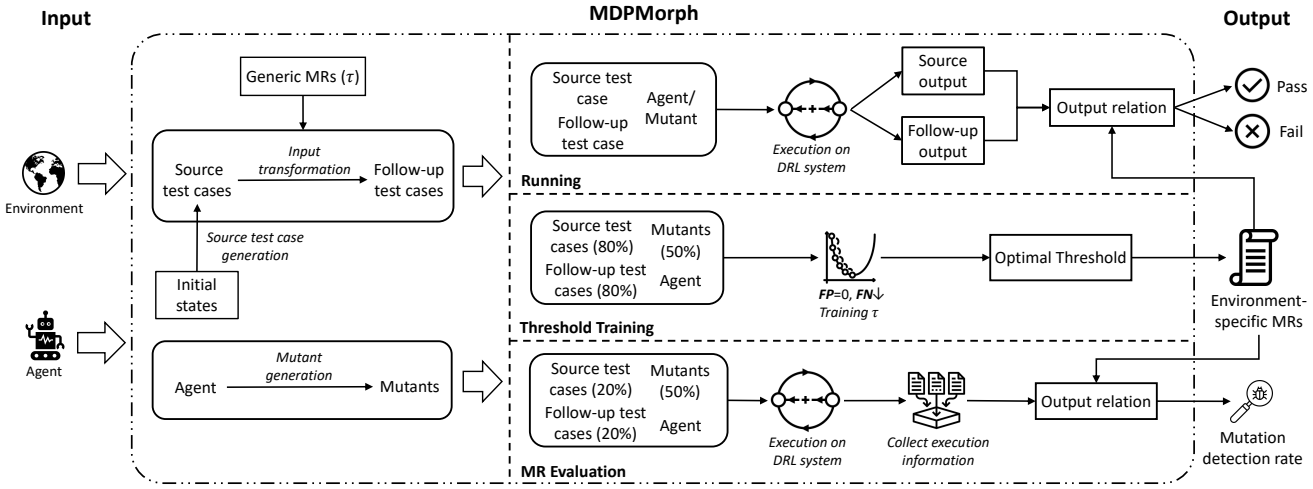

Figure 2. Logical architecture of MDPMORPH

When the input relation $\mathcal{R}_i$ holds true for a set of inputs, the corresponding output relation $\mathcal{R}_o$ is expected to hold true for the resulting outputs. An MR thus serves directly as a metamorphic test oracle.

Definition 2: Given an MR ($\mathcal{R}_i \Rightarrow \mathcal{R}_o$), a **metamorphic test oracle** is an executable Boolean expression that signals a fault if the input relation $\mathcal{R}_i$ holds true for a specific input set, but the output relation $\mathcal{R}_o$ does not.

Metamorphic test pairs (or groups) are typically created by first generating a test input (the **source input**), and then applying a transformation to produce additional inputs (the **follow-up test inputs**) that satisfy a specified input relation.

Definition 3: Given an MR ($\mathcal{R}_i \Rightarrow \mathcal{R}_o$) and a test input x_1, a **(metamorphic) input transformation** generates a new input x_2 such that $\mathcal{R}_i(x_1, x_2) = \text{true}$

MT executes the system under test on these pairs (or sets) of inputs, reporting an oracle violation whenever the output relation does not hold.

III. MDPMORPH

To alleviate the challenges outlined in Section I, we propose MDPMORPH, a novel MT framework specifically crafted for DRL agents. This framework is based on MDPs and focuses on the agent's core reasoning properties to automatically uncover faults. Figure 2 provides an overview of MDPMORPH, which consists of three modes: *Running*, *Threshold Training* and *MR Evaluation*. In the *Running* mode, potential faults in the agent are uncovered by comparing the execution outputs of automatically-generated source and follow-up test cases to verify whether the output relation of the MR is satisfied. The *Threshold Training* mode is used to optimize MR thresholds by analyzing the execution results of the original agent and its mutants. The *MR Evaluation* mode calculates the mutant detection rate to evaluate MR effectiveness. The workflow for these modes is outlined as follows.

Running Mode: The objective of this mode is to compare the execution results of source test cases and their corresponding follow-up test cases on the DRL system, and to output the pass or fail outcomes. First, it automatically generates source test cases by randomly sampling the initial states from the environment. From such test cases, it derives follow-up test cases using the input transformations defined by the MRs. It then executes the source and follow-up test cases on the same DRL model under test, producing the corresponding source and follow-up outputs. Finally, it compares the two outputs to verify whether they satisfy the output relation defined by the MR. Test cases that satisfy the expected relation are marked as "Pass", and those that do not as "Fail". Our proposed MRs use thresholds on the difference or similarity between source and follow-up outputs to determine whether the output relation is satisfied. A key factor in the effectiveness of our proposed MRs is the selection of appropriate threshold values, which are often sensitive to the environment under test. A MDPMORPH user can either define these thresholds based on domain knowledge or train them using the mode described below.

Threshold Training Mode: The goal of this mode is to automatically adapt generic MRs (applicable to any DRL system that satisfies the assumptions) into environment-specific MRs. The output relations defined in the generic MRs include multiple thresholds that specify acceptable deviation margins (see τ in Section III-B). These thresholds help prevent misjudgments caused by environmental noise, ensuring the MRs remain broadly applicable across different DRL systems.

In this mode, many different initial states are first extracted from the environment, serving as source test cases. These tests then undergo input transformations defined by the generic MRs to generate follow-up test cases. For mutant generation, our current implementation employs DEEPMUTATION [26], which produces different mutants by applying modifications to the architecture or parameters of the original agent. MDPMORPH selects 80% of the source and follow-up test cases, along with 50% of the mutants, for threshold training.

It then uses the false positives (FPs) and false negatives (FNs) identified by running each of the selected metamorphic test pairs [19], [20], [27], [28] to train the thresholds in the generic MRs. An FP occurs when, for a metamorphic test pair, the original agent behaves correctly, but the oracle mistakenly reports it as a failure [29]. An FN arises when, for a metamorphic test pair, a mutant exhibits erroneous behavior, yet the oracle incorrectly passes [29]. MDPMORPH then uses stochastic gradient descent to optimize a loss function involving FP and FN, based on the execution results of the source and follow-up test cases on both the original agent and its mutants. The goal is to ensure zero FP while minimizing FN [19], [20], [27]. This process ultimately gives the optimal threshold for each generic MR within the environment, enabling its transformation into an environment-specific MR.

Because this training overfits the current version, these thresholds can only be used in regression testing mode to detect faults in future versions of the agent [21]. Indeed, this mode assumes that the given model is correct. This approach aligns with previous studies that generate program assertions or MRs by minimizing FPs and FNs [19], [20], [27], all of which also operate under the regression testing assumption.

MR Evaluation Mode: The primary objective of this mode is to assess the effectiveness of the MRs. Although it uses the same methods for generating test cases and mutants as in the *Threshold Training* mode, the specific test cases and mutants used here are different. It executes the remaining 20% of the source and follow-up test cases on DRL systems containing the remaining 50% of the mutants. It then compares the resultant data with the expected output relations defined by the environment-specific MRs to determine whether each mutant is killed. Finally, it calculates the mutation detection rate for each MR.

The following sections formally define the MRs used in MDPMORPH, along with their underlying assumptions and definitions. Table I provides a summary of the key notations.

A. Core Assumptions and Definitions

We summarize the assumptions and definitions related to MDPs, which serve as the necessary prerequisites to ensure that the generic MRs in MDPMORPH satisfy their essential properties. These assumptions and definitions have already been applied in testing scenarios related to DRL systems [11], [9], [30], [31], [32].

Assumption 1: The MDP is Lipschitz continuous.

This assumption is based on the concept of Lipschitz continuity, which is defined as follows.

Definition 4: Given two metric sets (X, d_X) and (Y, d_Y), where d_X and d_Y denote the corresponding distance metrics, a function $f : X \to Y$ is said to be L-Lipschitz continuous if: $\forall (x_1, x_2) \in X^2, d_Y(f(x_1) - f(x_2)) \leq L d_X(x_1 - x_2)$

Several studies have explored the constraints imposed by Lipschitz continuity on transition and reward functions, providing a foundation for the study of Lipschitz continuity in MDPs [33], [34], [35]. We define the Lipschitz continuity of MDPs by referring to these works.

Table I
KEY NOTATIONS

Symbol	Description
$\mathcal{S}$, $\mathcal{A}_c$, $\mathcal{A}_d$	State space, continuous action space, discrete action space
s_t, a_t, r_t	State, action and reward at timestep t.
$P(s_t, a_t)$	State transition function at timestep t.
$R(s_t, a_t)$	Reward function at timestep t.
$\pi(s_t)$	The agent's policy at timestep t.
L_p	Lipschitz constant of state transition function.
L_r	Lipschitz constant of reward function.
s_t^1, a_t^1, r_t^1	The state, action, and reward at timestep t during the execution of the source test case.
S^1, R^1	The state sequence and reward sequence of a reasoning episode during the execution of the source test case.
s_t^2, a_t^2, r_t^2	The state, action, and reward at timestep t during the execution of the follow-up test case.
S^2, R^2	The state sequence and reward sequence of a reasoning episode during the execution of the follow-up test case.
τ	The threshold in MRs.
f	The total number of timesteps under a reasoning episode.

Definition 5: An MDP is Lipschitz continuous if both its state transition function P and its reward function R satisfy Lipschitz continuous. Specifically, there exists a Lipschitz constant L_p such that $||P(s,a) - P(s',a')|| \leq L_p(||s - s'|| + ||a - a'||)$ $\forall s, s' \in \mathcal{S}$ and $a, a' \in \mathcal{A}_c$ or $||P(s,a) - P(s',a)|| \leq L_p||s - s'||$ $\forall s, s' \in \mathcal{S}$ and $a \in \mathcal{A}_d$. Moreover, there exists a Lipschitz constant L_r such that $||R(s,a) - R(s',a')|| \leq L_r(||s - s'|| + ||a - a'||)$ $\forall s, s' \in \mathcal{S}$ and $\forall a, a' \in \mathcal{A}_c$ or $||R(s,a) - R(s',a)|| \leq L_r||s - s'||$ $\forall s, s' \in \mathcal{S}$ and $\forall a \in \mathcal{A}_d$. Here, $||\cdot||$ denotes the Euclidean distance.

A similar MDP Lipschitz continuity assumption on environment dynamics is commonly used in previous works analyzing the theoretical guarantees of DRL algorithms [36], [37], [38], [34], [39]. Moreover, it is a relatively mild assumption for practical applications [40], [41].

Assumption 2: The agent is memoryless and deterministic.

An agent is memoryless if it makes decisions only based on the current state, without relying on any past information. An agent is deterministic if, given the same state, the agent consistently selects the same action. These are inherently related to policies, forming the core of MDP research [42], [43]. Additionally, this assumption distinguishes between violations of MRs caused by errors in the agent's policy logic and those caused by internal randomness.

Assumption 3: The noise in the state transition process is the sole source of randomness in the agent's reasoning process.

We assume that the noise variables in the state transition process during the agent's reasoning are independent and remain fixed throughout the sequence. At the same time, as part of the environment, they are the sole source of randomness in the environment. This implicitly assumes that the state transitions are stationary and there are no unobserved

confounders [44], [45]. This assumption allows us to account for the randomness in the environment in designing MRs.

The three assumptions above define the applicability scope of the generic MRs in MDPMORPH. The following introduces the relevant definition used by generic MRs.

Definition 6: Let $V_\pi(s)$ be the state-value function under policy π. The optimal policy π^* is defined as $argmax_\pi V_\pi(s)$. That is, π^* attains the highest possible expected total reward from every state [46], [47].

B. Generic Metamorphic Relations (MRs)

This section presents the MRs currently implemented in MDPMORPH. Following previous work in the context of DRL systems, we consider a test case x to represent the initial state of the environment s_0 [9], [12], [48], [49], [50]. We use s_0^1 and s_0^2 to denote the source and follow-up test cases, respectively. By considering only the initial state and leaving all subsequent states unchanged, we ensure the generated executions remain feasible. Indeed, subsequent states are determined by the environment and the actions of the agent. The former cannot be easily altered without creating infeasible executions, and the latter is what we aim to test. The output $f(x)$ includes essential elements of a DRL execution, such as states, rewards, and actions, which collectively reflect the agent's decision-making.

To support MDPMORPH, we propose an MR design method tailored to the reasoning process of DRL agents. Since the MDP precisely defines how an agent selects actions from states, obtains rewards, and transitions between states over time, forming a sequence of "state-action-reward" steps. Leveraging the temporal structure of MDPs, our method decomposes the agent's reasoning process into two levels: local MDP and global MDP. Local MDP MRs are related to the agent's behavior within a selected local MDP region, emphasizing the local stability and consistency of the its decision-making. In contrast, global MDP MRs focus on the entire reasoning sequence of the agent to evaluate its overall performance.

Furthermore, local MDP can be further divided into single-step and multi-step. Single-step MRs examine immediate, instantaneous behavioral changes within a single step, while multi-step MRs assess the agent's performance and long-term behavior over multiple steps. This method encompasses MRs that span the agent's reasoning process from momentary decisions to long-term strategies. It enables a comprehensive capture of behavioral changes under different states and decision-making scenarios. Using this method, we propose nine generic MRs under certain assumptions and definitions of the MDP outlined in Section III-A. By leveraging Euclidean distance to assess the consistency of outputs, they can verify whether the interaction between the agent and the environment aligns with the intended objectives. We provide both descriptive explanations and formal representations of these MRs. We also provide a specific example for each MR based on an autonomous driving scenario to intuitively demonstrate the applicability and rationale of these MRs in real-world contexts. Due to space constraints, the detailed proofs can be found in the appendix [22].

Local MDP - single-step

MR 1: If the follow-up test case s_0^2 is derived from the source test case s_0^1 through a slight variation (perturbation δ_1, translation γ_1, or scaling λ_1), then, for continuous action spaces, the distance between the action values output by the agent should be less than a threshold τ_1; and, for discrete action spaces, their action values output by the agent should be identical. For the slight variations (δ_{1d}, γ_{1d}, or λ_{1d}) in input transformations within discrete action MRs, we referred to previous research [51], [52] and constrained the variation magnitude within 0.02, which has negligible impact on the agent's performance.

	Continuous Action Space	**Discrete Action Space**
MR1.1	$s_0^2 = s_0^1 + \delta_{1c} \implies$ $\|\|a_0^1 - a_0^2\|\| < \tau_{1.1}$	$s_0^2 = s_0^1 + \delta_{1d} \implies$ $a_0^1 = a_0^2$
MR1.2	$s_0^2 = s_0^1 + \gamma_{1c} \implies$ $\|\|a_0^1 - a_0^2\|\| < \tau_{1.2}$	$s_0^2 = s_0^1 + \gamma_{1d} \implies$ $a_0^1 = a_0^2$
MR1.3	$s_0^2 = \lambda_{1c} s_0^1 \implies$ $\|\|a_0^1 - a_0^2\|\| < \tau_{1.3}$	$s_0^2 = \lambda_{1d} s_0^1 \implies$ $a_0^1 = a_0^2$

Where δ_1, γ_1, and λ_1 represent noise perturbations, constant vectors, and constants respectively.

Example: If the vehicle's initial position is slightly changed, a well-trained agent should produce correspondingly minor adjustments to the vehicle's steering angle and throttle, with response magnitudes remaining within a predefined threshold.

Local MDP - multi-step

MR 2: If the source test case s_0^1 and the follow-up test case s_0^2 are identical, then after i steps and j steps respectively ($i > j$), the difference between the cumulative rewards of the source and follow-up test cases should exceed a threshold τ_2.

MR2	$\exists i = 0 \cdots n, j = 0 \cdots m, s_0^1 \equiv s_0^2 \wedge i > j \implies$ $\sum_{k=0}^{i} r_k^1 - \sum_{k=0}^{j} r_k^2 > \tau_2$

Example: If the vehicle, tasked with lane keeping, starts from the same lane position and speed, the difference between the rewards accumulated over a 60 seconds driving period and those accumulated during the first 10 seconds should exceed a predefined threshold.

MR 3: If the follow-up test case s_0^2 is derived from the source test case s_0^1 through a slight variation (perturbation δ_3, translation γ_3, or scaling λ_3), then there exists a time step k such that the distance between the last k steps of the state sequence from the source test case and the last k steps of the state sequence from the follow-up test case is less than a threshold τ_3.

Example: If the vehicle's initial position is slightly changed, the final trajectory states of the vehicle, such as position and speed in the last few steps, should remain highly consistent with the original trajectory, with differences staying within a predefined threshold.

MR3.1	$\exists k \geq 0, s_0^2 = s_0^1 + \delta_3 \implies D(\{s_{f-k}^1, s_{f-k+1}^1 ... s_f^1\}, \{s_{f-k}^2, s_{f-k+1}^2 ... s_f^2\}) < \tau_{3.1}$
MR3.2	$\exists k \geq 0, s_0^2 = s_0^1 + \gamma_3 \implies D(\{s_{f-k}^1, s_{f-k+1}^1 ... s_f^1\}, \{s_{f-k}^2, s_{f-k+1}^2 ... s_f^2\}) < \tau_{3.2}$
MR3.3	$\exists k \geq 0, s_0^2 = \lambda_3 s_0^1 \implies D(\{s_{f-k}^1, s_{f-k+1}^1 ... s_f^1\}, \{s_{f-k}^2, s_{f-k+1}^2 ... s_f^2\}) < \tau_{3.3}$

Where δ_3, γ_3, and λ_3 represent noise perturbations, constant vectors, and constants respectively.

Global MDP - entire reasoning sequence

MR 4: If the follow-up test case s_0^2 is identical to that of the source test case s_0^1, then the difference between the state sequences generated by the system from these test cases (measured by the distance function) should be within a threshold τ_4.

MR4	$s_0^2 \equiv s_0^1 \implies D(S^1, S^2) < \tau_4$

Example: If the vehicle starts from the same lane position and speed, the differences in its state trajectories in terms of position, velocity, and direction between two repeated driving sessions should remain within a predefined threshold.

MR 5: If the follow-up test case s_0^2 is identical to that of the source test case s_0^1, then the difference between the reward sequences generated by the system from these test cases (measured by the distance function) should be within a threshold τ_5.

MR5	$s_0^2 \equiv s_0^1 \implies D(R^1, R^2) < \tau_5$

Example: If the vehicle starts from the same lane position and speed, the differences in its reward trajectories, including lane keeping rewards and collision penalties, between two repeated driving sessions should remain within a predefined threshold.

IV. Evaluation

The primary objective of our experiments is to address the following three research questions.

RQ1 - Assumptions Validation. *To what extent are the assumptions satisfied by the DRL systems under test?*

RQ2 - Effectiveness. *What is the mutation detection rate of our proposed MRs?*

RQ3 - Mutants Inspection. *What are the characteristics and underlying causes of the mutants that are undetected by each MR?*

RQ1 examines whether the assumption conditions of MDPMorph hold in the DRL systems we use, as these conditions form the basis for applying its MRs. To achieve this, we performed a thorough inspection of the source code and theoretically validated these assumptions through formal derivations. These inspections are not strictly necessary, as MDPMorph can still effectively detect faults even if the assumptions are not met, although the lack of theoretical guarantees may result in false positives (FPs).

RQ2 evaluates the effectiveness of the MRs in MDPMorph. We generated mutants and diverse metamorphic test pairs, trained thresholds to adapt generic MRs to each environment, and then ran the test cases on mutated DRL systems to measure mutation detection rates and perform statistical analysis.

RQ3 analyses the mutants that were not detected by each individual MR. We investigate their characteristics and the underlying causes, aiming to provide developers insight into these types of faults.

A. Subjects

The DRL environments under test are CartPole, LunarLander, and BipedalWalker. These are all part of a well-known open-source gymnasium toolkit[2] developed by OpenAI, which has been widely used in testing DRL systems [10], [31], [53], [12], [49]. Figure 3 shows 2D visual representations of the environments.

CartPole is a pole-balancing control task. It involves controlling a cart that moves horizontally along a track to prevent an inverted pole, hinged to its top, from falling over. The state vector consists of the cart's position and velocity, as well as the pole's angle and angular velocity. At each time step, the agent selects one of two discrete actions: applying a fixed force to either the left or the right. A reward of 1 is given for every time step the pole remains within an upright threshold. The episode terminates if the pole falls beyond this threshold, or if a predefined number of steps is reached.

LunarLander is a classic rocket trajectory optimization problem involving a lander. The state includes information such as the lander's position, velocity, and whether both legs are in contact with the ground. The action space is composed of two continuously varying thrusts: one for the main engine and the other for the lateral boosters. Rewards are provided based on engine ignition timing during landing and the final landing status. The episode terminates if the lander crashes, moves out of the viewport, or fails to land properly.

BipedalWalker involves controlling a two-legged robot that must traverse a rugged terrain. The walker features articulated legs with multiple joints, and the state of the environment comprises details such as the robot's position, velocity, joint angles, and foot contact sensors. The action space consists of the movement velocities of four joints—hips and knees—with their values continuously ranging within [-1, 1]. Rewards are provided for forward progress and efficient movement, while penalties are imposed for instability, excessive energy use, or falling. The episode terminates when the walker falls, leaves the designated path, or reaches the time limit.

OpenAI provides environments that define the dynamic properties of these three tasks and automatically generate data for training DRL models. To train our DRL models under test, we employed the actor-critic [54] and soft actor-critic [55] algorithms, as these methods are widely adopted by the DRL community. Specifically, we selected a well-tested, open-source implementation of these algorithms[3]. We terminated the training

[2]https://github.com/OpenAI/gym

[3]https://github.com/UoA-CARES/cares_reinforcement_learning

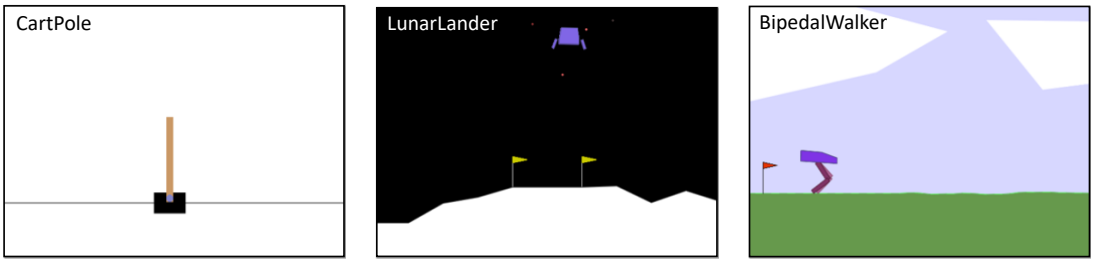

Figure 3. Environments of the DRL systems under test

Table II
THE MUTATION OPERATORS WE USED (FROM DEEPMUTATION [26])

Mutation Operator	Level	Description
Gaussian Fuzzing (GF)	Weight	Fuzz weight by Gaussian Distribution
Weight Shuffling (WS)	Neuron	Shuffle selected weights
Neu. Eff. Block. (NEB)	Neuron	Block a neuron effect on following layers
Neu. Activ. Inv. (NAI)	Neuron	Invert the activation status of a neuron
Neuron Switch (NS)	Neuron	Switch two neurons of the same layer
Layer Deactivation (LD)	Layer	Deactivate the effects of a layer
Layer Addition (LA)	Layer	Add a layer in neuron network
Act. Fun. Remov. (AFR)	Layer	Remove activation functions

Table III
COMPARISON OF TASK COMPLETION RATES BETWEEN THE ORIGINAL AGENTS AND 80 MUTANTS ACROSS THREE SUBJECTS

	CARTPOLE	LUNARLANDER	BIPEDALWALKER
Original agent (%)	95	96	90
Mutants med (%)	0	53	0
Mutants avg (%)	6.78	41.64	20.53
Mutants better or same (count)	4	0	5

process once the DRL agent successfully completed the task 100 times, ensuring that the three DRL models under test were sufficiently high-quality.

B. Experimental Setup

Test Case Generation. To evaluate MDPMORPH and the proposed MRs using the simplest test generation approach, we relied on random test generation. To ensure consistency and fairness, all subjects under test were evaluated within the same reachable and realistic state space. Accordingly, the source test cases were randomly generated from the range of initial states defined during the training of the agent under test. To account for the stochastic nature of random generation, we created 50 distinct test suites of source test cases for each environment, with each suite comprising 100 test cases. Of these, 40 suites were used to train the thresholds, while the remaining 10 suites were reserved to evaluate the effectiveness of the MRs.

Since we use initial states as test cases, we cannot directly control the environment's dynamics or the agent's behavior. Therefore, for MRs that specify input transformations based solely on initial states (i.e., MR1 and MR3), we derive follow-up test cases directly from the corresponding source test cases.

For MRs whose input relations depend on states other than the initial ones (i.e., MR2, MR4, and MR5), we select follow-up test cases from the existing pool of source test cases. We randomly examine pairs until we find one that satisfies the input relation defined by the MR, thereby ensuring proper alignment between source and follow-up test cases.

Mutant Generation. To generate DRL mutants, we used DEEPMUTATION [26], which provides source- and model-level mutation operators. for deep learning systems. However, due to key differences between traditional DL and DRL training, source-level mutations are not suitable for our context. Thus, we used only the model-level mutation operators. Table II lists the eight model-level operators provided by DEEPMUTATION, all of which were used in our experiments.

Recent mutation testing approaches for DRL systems [56], [53], [57] also use DEEPMUTATION's model-level mutations and introduce additional operators applied during DRL training. We chose not to include training-related mutants (e.g., limited training budget) in our experiments. This decision was motivated by two factors. First, retraining DRL models is computationally expensive: each retraining in our experiments required, on average, about 1.5 hours. At our experimental scale, even a moderate number of retrains becomes computationally infeasible. Second, the randomness in DRL training makes training-time mutations hard to reproduce reliably.

For the weight-level and neuron-level mutation operators in Table II, we adopt a randomized method [26] to modify either the original agent's weight matrices or the information within its neurons, thereby producing mutants. The implementation of layer-level mutation operators is more complex [58]. First, we analyze the overall structure of the agent to identify layers that meet the mutation conditions without disrupting the internal data flow. Based on this, we mutate these layers to construct a new mutant. We configured each mutation operator to randomly generate 20 distinct DRL mutants in each environment, resulting in a total of 160 mutants. A random sample of 80 was used for threshold training, and the remaining 80 for MR evaluation.

We evaluated the performance of the 80 mutants for MR evaluation to confirm that they introduced observable behavioral differences. Specifically, we generated 100 random test cases for each of the three subjects and calculated task completion rates under the default settings. Compared to the original well-trained agent, only 4, 0, and 5 mutants out of the 80 achieved the same or higher completion rates in CARTPOLE, LUNARLANDER, and BIPEDALWALKER, respectively. Table III shows the task

completion rates. It is worth noting that even if a mutant outperforms the original agent in terms of completion rates, it may still exhibit behavioral inconsistencies in certain states. For example, in autonomous driving, a slight change in the vehicle's initial position may cause drastic steering changes due to agent faults, yet the vehicle can still complete the task through subsequent self-adjustments. Therefore, considering all 80 mutants contributes to a more comprehensive evaluation of the MR's detection capability.

Experimental Procedures. As we lack the domain knowledge required to manually define thresholds (unlike the original developers of the OPENAI environments), we did not select the threshold values ourselves. Instead, we relied on a *Threshold Training* mode and thus evaluated our method in a regression testing context. Our experiments involved two main steps:

Step 1: Training thresholds. We first determined the threshold values for the nine generic MRs discussed in Section III-B across the three different environments. That is, we transformed these generic MRs into environment-specific MRs. To achieve this, we use *stochastic gradient descent* [59] to determine the optimal threshold for each combination of MR and environment (see Threshold training mode in Figure 2). Specifically, stochastic gradient descent first initializes the threshold of the MR to zero and then iteratively adjusts it by executing 4,000 test cases on the original agent: increasing it upon MR violations to relax constraints, and slightly decreasing it otherwise to tighten them, using a small learning rate (from 0.001 to 0.1). Additionally, we use extra mutants (80 for each environment) to validate the threshold values, aiming to maximize fault detection. Once the threshold values converge, we consider the final value as the optimized threshold for the MR.

Step 2: Evaluating MRs. Using the determined threshold, the original agent did not exhibit any violations of the MRs across all test cases (including the 20% reserved for evaluation), indicating no FPs. This enabled mutation testing. We then evaluated the performance of the MRs, focusing on their ability to detect mutants across the three different environments, as shown in the *MR Evaluation* mode in Figure 2. Specifically, a mutant is considered detected by an MR if the execution results of the source and follow-up test cases fail to satisfy the expected relation defined by that MR, as illustrated in the *Running* mode of Figure 2.

We conducted our experiments on a server equipped with an AMD® Ryzen Threadripper 3990X 64-Core Processor 2.9 GHz, an RTX® 6000 24 GiB, running Ubuntu 20.04 and Python 3.10.

Table IV reports the total time MDPMORPH takes to run all nine MRs, averaged per test suite. Although threshold training is done once per MR, it involves generating and running multiple test suites. Test generation was also repeated 10 times to account for randomness. As such, the table shows the total time for all nine MRs but averaged by test suite. On average, running and evaluating all MRs took 16 hours per test suite.

C. Evaluation Metrics

We use the **mutation detection rate** [60] to measure MR effectiveness, defined as the proportion of detected mutants: $M_{kr}/M_t - M_e$. Here, M_{kr} is the number of mutants detected by the test suite derived from the MR, M_t is the total number of mutants, and M_e is the number of equivalent mutants. Equivalent mutants refer to those who have been used for modification, yet produce no significant differences from the original agent during the reasoning process. If any metamorphic test pair on a mutant violates the MR, the test suite derived for that MR is considered to have detected (killed) the mutant. A higher mutation detection rate means the MR detects more mutants, showing greater effectiveness.

Table IV
AVERAGE TIME FOR THRESHOLD TRAINING AND FOR RUNNING/EVALUATION PER TEST SUITE AND FOR ALL MRS (APPROX.)

	Threshold Training	**Running and Evaluation**
CARTPOLE	3 minutes	2 minutes
LUNARLANDER	9 hours	7 hours
BIPEDALWALKER	10 hours	9 hours
Total	19 hours	16 hours

D. RQ1: Assumptions Validation

To answer RQ1, we thoroughly inspected the source code and conducted theoretical analysis to verify whether the systems used satisfy our proposed assumptions. While the environments mostly satisfy all our assumptions, we made minor changes (e.g., noise injection and signal filtering in the state and reward modeling processes). These changes ensure that the assumptions are always met during the testing process (the experiments for RQ2 and RQ3) without altering the core logic of the environments or models, and make them more representative of real-world conditions.

Assumption 1 *(Lipschitz continuity of MDP).* We extracted and analyzed the state transition functions and reward functions in all three environments, focusing only on continuous physical quantities and omitting abrupt indicators.

State transition functions:

State transition function of CARTPOLE*:*

$$s_{t+1} = s_t + \kappa g(s_t, a_t), \quad \text{where } g(s,a) = \begin{pmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{pmatrix} \tag{1}$$

where s represents the state information of the cart and the pole; κ is the time step; $\dot{x}$ and $\ddot{x}$ are the cart's velocity and acceleration; $\dot{\theta}$ and $\ddot{\theta}$ are the pole's angular velocity and angular acceleration. Since the computation of the continuous-time velocity field $g(s,a)$ involves only addition, multiplication, and trigonometric functions [61], and the state variables are always confined within a bounded set (as exceeding the bounds triggers termination), it follows that the state transition of CARTPOLE is Lipschitz continuous.

State transition function of LUNARLANDER*:*

$$s_{t+1} = P(s_t, a_t) \tag{2}$$

Table V
RQ2 - MUTATION DETECTION RATES FOR EACH MR ACROSS TEN TEST SUITE RUNS

MR	CARTPOLE				LUNARLANDER				BIPEDALWALKER			
	Min	Max	Med	Avg	Min	Max	Med	Avg	Min	Max	Med	Avg
MR1.1	0.76	0.79	0.78	0.77	0.86	0.89	0.88	0.88	0.91	0.96	0.94	0.94
MR1.2	0.76	0.81	0.79	0.77	0.75	0.86	0.85	0.84	0.74	0.83	0.80	0.79
MR1.3	0.73	0.76	0.74	0.74	0.85	0.86	0.85	0.86	0.86	0.91	0.88	0.88
MR2	0.67	0.69	0.68	0.68	0.91	0.98	0.96	0.96	0.69	0.73	0.71	0.70
MR3.1	0.58	0.61	0.60	0.59	1.00	1.00	1.00	1.00	0.90	0.93	0.91	0.91
MR3.2	0.61	0.64	0.63	0.63	1.00	1.00	1.00	1.00	0.88	0.91	0.90	0.90
MR3.3	0.51	0.58	0.55	0.54	1.00	1.00	1.00	1.00	0.90	0.93	0.91	0.91
MR4	0.84	0.90	0.85	0.87	0.69	0.95	0.81	0.83	0.76	0.84	0.81	0.80
MR5	0.95	0.98	0.96	0.96	0.95	0.98	0.96	0.97	0.95	0.96	0.96	0.96

where s denotes the normalized position, velocity, angle, and angular velocity; and a is the linear and angular torques from the main and side engines. This is composed of a series of operators (including sine, hyperbolic tangent, linear combination, and Box2D body dynamics [62]), all of which are continuously differentiable. Therefore, the state transition of LUNARLANDER is Lipschitz continuous.

State transition function of BIPEDALWALKER*:*

$$s_{t+1} = P(s_t, a_t) \tag{3}$$

where s includes the walker's position, velocity, angle, angular velocity, linear velocity, and joint states; and a is the motor speed values of the joints. The transition mapping in Equation 3 is composed of a combination and superposition of 1-Lipschitz functions (such as "clip" function) and continuously differentiable functions (such as mechanical equations and Euler integration). Therefore, the state transition of BIPEDALWALKER is Lipschitz continuous.

Reward functions:

Reward function of CARTPOLE*:*

$$R(s_t, a_t) = 1 \tag{4}$$

Since this is a constant function that outputs the same value for all inputs, it exhibits no variation across the entire state-action space and thus is Lipschitz continuous.

Reward function of LUNARLANDER*:*

$$R(s_t, a_t) = shaping(s_{t+1}) - shaping(s_t) - 0.3(m + s) \tag{5}$$

where $shaping(s) = -100(\sqrt{x^2 + y^2} + \sqrt{v_x^2 + v_y^2} + |\theta|)$; x and y denote the position of the lander; v_x and v_y represent its velocity components; θ indicates the lander's orientation angle; m represents the main engine power; and s denotes the side engine power. Since the square root function, the absolute value function, and discrete variables with finite changes are all Lipschitz continuous over a bounded domain, the overall shaping function is Lipschitz continuous within the bounded state space. Moreover, both the main engine and the side engine take continuous values within the range (-1, 1). Therefore, the reward function of LUNARLANDER is Lipschitz continuous.

Reward function of BIPEDALWALKER*:*

$$R(s_t, a_t) = shaping(s_{t+1}) - shaping(s_t) - \beta \sum_{i=1}^{k} min(|a_{t,i}|, 1) \tag{6}$$

where $shaping(s) = \frac{130x}{\alpha} - 5|\theta|$; x is the forward distance; θ is the tilt angle of the head; α and β are constants; and k is the number of actions. Since this consists solely of continuously differentiable linear terms, absolute values, and the summation of action values within a specified range (each term is Lipschitz continuous), the overall function is therefore also Lipschitz continuous.

Assumption 2 *(memoryless and deterministic of the agent).* According to the Markov property of DRL systems, the current state encapsulates all the information necessary for the agent's decision-making. As a result, the agent's decisions are memoryless. To achieve deterministic behavior of the agent during testing, we adopt a deterministic action selection strategy, which is implemented at the code level by replacing the potentially stochastic action sampling with always selecting the action with the highest output probability. This ensures that the agent's decisions remain consistent under identical states. Importantly, this modification does not alter the architecture or parameters of the trained model and thus preserves the original policy's expressive capacity.

Assumption 3 *(randomness of the environment).* Code inspection of LUNARLANDER showed that noise is introduced only through `wind_idx` and `torque_idx` during state transitions. Both undergo trigonometric transformation and remain within [–1, 1], satisfying Assumption 3. As CARTPOLE and BIPEDALWALKER lack inherent stochastic noise, we added bounded noise to their state transitions to meet Assumption 3.

RQ1 in summary: The three environments mostly satisfy our assumptions. We made small adjustments to ensure conformity during testing.

E. RQ2: Effectiveness

To address RQ2, we first train the thresholds of the generic MRs across the three environments, yielding environment-specific MRs. Subsequently, we test these MRs against mutants

generated by the mutation operators listed in Table II, using ten test suites, each comprising 100 test cases. Finally, following the evaluation metrics outlined in Section IV-C, we evaluate the effectiveness of the MRs across the three environments.

To calculate the mutation detection rate, we recorded the failure rate of each test suite derived from an MR when executed on mutants—that is, the proportion of test cases within the suite that successfully detected mutants. If the failure rate of a test suite exceeds zero, we consider that the test suite has detected the mutant. We observed that these test suites yielded a failure rate of zero when executed on the original agents across all three environments. When executed on the mutants, the average failure detection rates for each environment are as follows: In CARTPOLE, the average failure rate of MRs ranges from 0.22 (MR1.1) to 0.45 (MR5), with an overall average of 0.32. In LUNARLANDER, the range spans from 0.10 (MR4) to 0.51 (MR3.3), with an average of 0.31. For BIPEDALWALKER, the average failure rates vary from 0.05 (MR1.2) to 0.66 (MR5), with an average of 0.37. Overall, whether in relatively simple tasks like the cart pole problem or more complex scenarios such as bipedal walking, the proportion of test cases derived from MRs that successfully detect mutants remains relatively stable across the test suites. This indicates that the MRs we designed possess generalizability across different environments.

Based on these failure rates, we found that all mutants across the three environments are detected by the test suites derived from the MRs. This indicates that none of the mutants are equivalent. Therefore, all mutants were taken into account when calculating the mutation detection rate. This avoided the challenge of detecting equivalent mutants.

Additionally, the proportion of the same mutants detected by all test suites derived from all MRs accounts for 62% of the total number of mutants, demonstrating the powerful mutation detection capability of the MRs.

Table V shows the mutation detection rates of each MR across the ten test suites. Columns "Min", "Max", "Med" and "Avg" represent the minimum, maximum, median and average mutation detection rates across the ten test suites, respectively. The results show that the MRs based on the global MDP are the most effective. MR5 in particular achieves a minimum mutation detection rate above 0.95 across all three environments, with minimal min/max variation, indicating it is a highly effective MR. The single-step MRs in the local MDP also achieved relatively high mutation detection rates; however, there was a noticeable degree of fluctuation in the BIPEDALWALKER, indicating that the mutants in this environment are more sensitive to input variations. Finally, the mutation detection rates of the multi-step MRs in the local MDP exhibited significant variability across different environments.

For the environments, the average mutant detection rates in the CARTPOLE, LUNARLANDER, and BIPEDALWALKER are 0.73, 0.93, and 0.87, respectively. Specifically, the MRs exhibit the highest detection rates in the LUNARLANDER environment, with all median detection rates above 0.81 and three MRs even reaching 1.0. The detection rates in BIPEDALWALKER are also relatively high, with most MRs showing median values around 0.9. In contrast, the detection rates in CARTPOLE vary widely, ranging from 0.55 to 0.96. This variation may be attributed to the nature of decision-making in complex control tasks, where agents must coordinate multiple submodules within the environment. Any minor fault in the decision-making process can trigger a cascading amplification effect in the MDP, manifesting in various forms. The MRs we designed are capable of effectively capturing these nuanced differences. In simpler scenarios, however, the agent's decision-making tends to rely on more singular factors, and only certain MRs are directly relevant to the core decisions in such environments. As a result, the detection rates fluctuate more significantly.

Table VI
RQ3 - THE NUMBER OF MUTANTS AT DIFFERENT LEVELS THAT CAN BE DETECTED BY EXACTLY n MRS ($n = 1, 2, \ldots, 9$)

	1	2	3	4	5	6	7	8	9
Weight-level	0	0	0	0	2	3	2	5	18
Neuron-level	7	0	2	4	3	2	8	15	79
Layer-level	0	2	11	1	2	12	2	8	52
Total	7	2	13	5	7	17	12	28	149

RQ2 in summary: All mutants under MDPMORPH were detected at least once by the test suites derived from these MRs. These MRs also achieved an average mutant detection rate of 0.84 and demonstrated robust detection performance in complex environments.

F. RQ3: Mutants Inspection

To address RQ3, we analyzed the detection results for all 240 generated mutants across the three environments and all ten test suites, focusing on which mutants were not detected by each individual MR. These undetected mutants were then categorized according to the three mutation levels: weight, neuron, and layer (see Table II).

Table VI summarizes the number of mutants at different levels that can be detected by exactly n ($n = 1, 2, \ldots, 9$) MRs (considering all ten test suites). The results show that weight-level mutants are relatively easy to detect, with five MRs being sufficient to detect all weight-level mutants. In contrast, neuron-level and layer-level mutants are more difficult to detect, with some mutants being detectable by only one or two MRs.

When considering each test suite individually, the results are quite consistent: different test suites generated for the same MRs detected exactly the same mutants. The only exception is the BIPEDALWALKER, where two neuron-level mutants went undetected by some suites.

In addition, we also recorded the distribution of undetected mutants across different levels under each MR. Figure 4 illustrates the undetected rates of these levels across different MRs. It shows that, within the single-step of the local MDP, the undetected rate for neuron-level and layer-level mutants was significantly higher than that for weight-level mutants. The root cause is that single-step state-action mapping validation uses

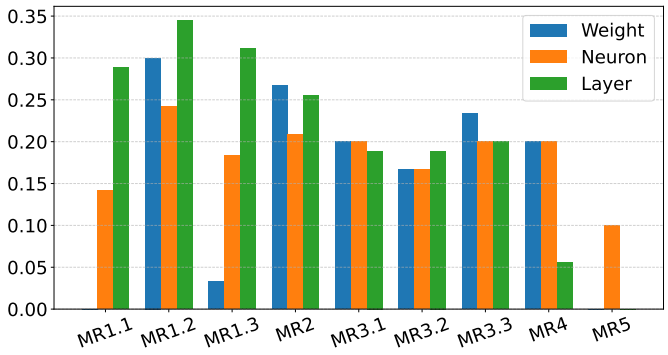

Figure 4. RQ3 - The level distribution of undetected mutants under each MR

only generic initial test cases, often failing to activate specific neurons or entire layers. Typical state inputs may fail to activate critical paths when mutations alter only a neuron's activation function or replace an entire layer's mapping. As a result, the output differences remain subtle and fall below the detection threshold defined by MR1. For the multi-step MRs of the local MDP, the undetected rate remains roughly consistent across weight-level, neuron-level, and layer-level mutations. Extending the validation window to fixed-length trajectories causes all mutation types (parameter tweaks, single-neuron failures, or layer-level alterations) to accumulate similar deviations in decision sequences. Over several steps, such deviations tend to accumulate, resulting in significant differences in the final state sequences or rewards. Since the same distance metrics and threshold criteria are applied to all three mutation levels, their undetected rates naturally converge. Furthermore, we find that global MRs exhibit strong detection capabilities and capture a broader range of mutant types. However, MR4 shows a higher proportion of undetected mutants compared to MR5. This is because the state space relied upon by MR4 is typically high-dimensional, and the source test cases fail to sufficiently trigger its detection of subtle anomalies. In contrast, MR5 leverages cumulative rewards, allowing even minor anomalies to be aggregated into a single value, thereby enabling it to detect nearly all mutants.

These results reveal that, when constructing test cases, it would be beneficial to adopt activation maximization techniques to trigger potential mutations at the neuron or layer level. Moreover, future work should generate test cases capable of covering a broader range of states in high-dimensional state spaces, thereby compensating for the limited sensitivity to subtle anomalies.

RQ3 in summary: Compared with weight-level mutants, neuron-level and layer-level mutants are more difficult to detect. This suggests that greater attention should be given to strategies such as maximizing neuron activation in the construction of test cases.

V. Threats to Validity

Construct. The mutation detection rate used to evaluate the effectiveness of MRs is based on a statistical comparison of agent performance, following the method proposed by Hu et.al [60]. While our evaluation metric provides a principled way to quantify behavioral differences, it relies on threshold-based distinctions, which may not capture all the differences between the original and mutant agents. Moreover, the configuration space of mutation operators introduces variability in how mutant detection is defined. Therefore, the mutation detection rate may be sensitive to the choice of metrics, thresholds, and operator configurations, potentially affecting the accuracy of MR effectiveness assessments.

External. A potential threat to the external validity is the generalization of the results. We mitigated this threat by carefully selecting three widely used subjects for validation ensuring that they cover diverse tasks, action space types, and application domains. These are all part of the well-known open-source gymnasium toolkit developed by OpenAI.

VI. Related Work

This section discusses related work divided into two main groups: testing for DRL systems and MT.

Testing for Deep Reinforcement Learning (DRL) systems primarily involves search-based testing [10] and fuzzing techniques [9], [50], [49]. To highlight some representative techniques: Zolfagharian et al. [10] introduced STARLA, which leverages genetic algorithms to efficiently identify policy failures and extract failure rules for safety risk assessment. Pang et al. [9] proposed MDPFuzz, a black-box MDP fuzzing framework that uses Gaussian mixture models and local sensitivity to discover state sequences that lead to crashes. Wang et al. [49] proposed SeqDivFuzz, a black-box testing framework for sequence decision problems modeled as MDPs. Wan et al. [50] proposed DRLFuzz, a coverage-guided fuzz testing framework specifically designed for DRL systems, addressing issues such as environmental distribution biases during DRL system training.

Similarly, MDPMorph also uses MDP models, takes initial states as test inputs, and shares some of the same assumptions and the general goal of exposing bugs in DRL agents. However, these techniques all rely on human-written oracles. In contrast, MDPMorph employs MT to automate the oracle problem, thereby eliminating the need for manual oracle construction.

Metamorphic Testing (MT) [14] has seen significant advances in both traditional [15], [16] and ML-based software systems [17], [18], [63], [64], [65]. In the context of traditional software, recent techniques [19], [20], [21] have explored the automated generation of MRs using search-based methods and evolutionary algorithms, guided by the minimization of false positives and/or false negatives (the latter obtained through mutation testing like MDPMorph). MDPMorph draws inspiration from these approaches, not to generate new MRs, but to optimize the threshold values of our proposed MRs.

Although MT has been extensively applied to testing ML systems [16], existing MT methods and MRs are not specifically designed for RL or DRL systems. In particular, RL systems involve unique elements such as policies, reward functions, and state transitions, which require tailored MRs and testing approaches [11]. To the best of our knowledge, the work by Eniser et al. [11] is the most closely related to ours and the only existing approach that applies MT for action-policy testing in RL. They proposed a relaxed MR that permits minor policy deviations under environment variations to detect decision inconsistencies. Our work differs substantially in several key aspects: (i) we introduce nine generic MRs designed across multiple semantic levels, (ii) we present a method to automatically instantiate these MRs in specific environments via threshold training, and (iii) our framework automatically generates test cases, enabling end-to-end automation of metamorphic testing for DRL systems. Nonetheless, our MR2 is similar in nature to the one they proposed, so we did not consider a comparison necessary. In addition, our approach is based on MDP, ensuring that the MRs reflect fundamental properties of agent behavior.

VII. Conclusion

This paper presented MDPMorph, a Metamorphic Testing (MT) framework for Deep Reinforcement Learning (DRL) agents. Grounded in the theoretical foundations of Markov Decision Processes (MDPs), MDPMorph introduces a systematic approach to designing Metamorphic Relations (MRs) specifically tailored for DRL agents. Based on this approach, we propose nine MRs, whose necessity properties are theoretically proven (see Appendix [22]). Our experimental results on three popular OpenAI environments demonstrate that all generated mutants were successfully detected by at least one of our MRs.

Our nine proposed MRs do not aim to be a comprehensive set of MRs that capture all characteristics of DRL systems. Ensuring the completeness of a set of MRs remains a well-known open challenge in the field. However, we believe they capture the essence of DRL reasonably well because they are considering both local and global MDP and they predicate on actions, rewards, and states, which are specific to DRL systems. We hope that our work will inspire the research community to define additional MRs to capture the characteristics of DRL systems more comprehensively.

This work is among the first to explore MT for DRL. Based on our findings, we identify four promising directions for future research: First, based on the results of RQ3, techniques like maximizing neuron activation could be explored to generate test cases more effectively than random methods. Second, designing new MRs tailored to DRL systems remains an important area for future work. Third, our high average failure rate (i.e., number of failing tests) suggests that future work could reduce MDPMorph's computational cost by generating fewer test cases and evaluating the impact on fault detection. Fourth, extending MDPMorph to support a broader range of RL systems (not limited to DRL) would allow us to evaluate its effectiveness in a more general RL context.

Acknowledgements

This work was supported in part by the National Key R&D Program of China under Grant 2024YFB3311503, the National Natural Science Foundation of China under Grant 62372021, and the International Joint Doctoral Education Fund of Beihang University.

References

[1] A. K. Shakya, G. Pillai, and S. Chakrabarty, "Reinforcement learning algorithms: A brief survey," *Expert Systems with Applications*, vol. 231, p. 120495, 2023.

[2] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2*. Springer, 2018, pp. 426–440.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] C.-J. Hoel, K. Wolff, and L. Laine, "Ensemble quantile networks: Uncertainty-aware reinforcement learning with applications in autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 6, pp. 6030–6041, 2023.

[5] Z. Huang, H. Liu, J. Wu, and C. Lv, "Conditional predictive behavior planning with inverse reinforcement learning for human-like autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 7244–7258, 2023.

[6] D. Valencia, H. Williams, Y. Xing, T. Gee, M. Liarokapis, and B. A. MacDonald, "Image-based deep reinforcement learning with intrinsically motivated stimuli: On the execution of complex robotic tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2024, pp. 587–594.

[7] D. Chen, M. R. Hajidavalloo, Z. Li, K. Chen, Y. Wang, L. Jiang, and Y. Wang, "Deep multi-agent reinforcement learning for highway on-ramp merging in mixed traffic," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 11, pp. 11 623–11 638, 2023.

[8] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *arXiv preprint arXiv:1704.02532*, 2017.

[9] Q. Pang, Y. Yuan, and S. Wang, "MDPFuzz: testing models solving markov decision processes," in *ACM International Symposium on Software Testing and Analysis*, 2022, pp. 378–390.

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

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

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

[13] A. Sunba, J. Hassine, and M. Ahmed, "Testing reinforcement learning systems: A comprehensive review," *Journal of Systems and Software*, 2025.

[14] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," 1998.

[15] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[16] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–27, 2018.

[17] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[18] F. U. Rehman and M. Srinivasan, "Metamorphic testing for machine learning: Applicability, challenges, and research opportunities," in *IEEE International Conference On Artificial Intelligence Testing*, 2023, pp. 34–39.

[19] J. Ayerdi, V. Terragni, A. Arrieta, P. Tonella, G. Sagardui, and M. Arratibel, "Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1264–1274.

[20] J. Ayerdi, V. Terragni, G. Jahangirova, A. Arrieta, and P. Tonella, "GenMorph: Automatically generating metamorphic relations via genetic programming," *IEEE Transactions on Software Engineering*, vol. 50, no. 7, pp. 1888–1900, 2024.

[21] J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei, "Search-based inference of polynomial metamorphic relations," in *ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 701–712.

[22] J. Li, Z. Zheng, Y. Xing, D. Ren, S. Cho, and V. Terragni, "Open source implementation of MDPMorph," https://github.com/tissten/MDPMorph, 2025.

[23] ——, "MDPMorph: experimental data," https://doi.org/10.5281/zenodo.16908139, 2025.

[24] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[25] E. A. Feinberg and A. Shwartz, *Handbook of Markov decision processes: methods and applications*. Springer Science & Business Media, 2012, vol. 40.

[26] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *IEEE International Symposium on Software Reliability Engineering*, 2018, pp. 100–111.

[27] V. Terragni, G. Jahangirova, P. Tonella, and M. Pezzè, "Evolutionary improvement of assertion oracles," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1178–1189.

[28] V. Terragni, G. Jahangirova, P. Tonella, and M. Pezzè, "GAssert: A fully automated tool to improve assertion oracles," in *IEEE/ACM International Conference on Software Engineering Companion*, 2021, pp. 85–88.

[29] G. Jahangirova, D. Clark, M. Harman, and P. Tonella, "Test oracle assessment and improvement," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2016, p. 247–258.

[30] M. Tappler, F. C. Córdoba, B. K. Aichernig, and B. Könighofer, "Search-based testing of reinforcement learning," *arXiv preprint arXiv:2205.04887*, 2022.

[31] A. Zolfagharian, M. Abdellatif, L. C. Briand *et al.*, "SMARLA: A safety monitoring approach for deep reinforcement learning agents," *IEEE Transactions on Software Engineering*, vol. 51, no. 1, pp. 82–105, 2025.

[32] Q. Mazouni, H. Spieker, A. Gotlieb, and M. Acher, "Testing for fault diversity in reinforcement learning," in *ACM/IEEE International Conference on Automation of Software Test*, 2024, pp. 136–146.

[33] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare, "Deepmdp: Learning continuous latent space models for representation learning," in *International Conference on Machine Learning*, 2019, pp. 2170–2179.

[34] K. Asadi, D. Misra, and M. Littman, "Lipschitz continuity in model-based reinforcement learning," in *International Conference on Machine Learning*, 2018, pp. 264–273.

[35] K. Hinderer, "Lipschitz continuity of value functions in markovian decision processes," *Mathematical Methods of Operations Research*, vol. 62, pp. 3–22, 2005.

[36] O. Gottesman, K. Asadi, C. Allen, S. Lobel, G. Konidaris, and M. Littman, "Coarse-grained smoothness for rl in metric spaces," *arXiv preprint arXiv:2110.12276*, 2021.

[37] C. Le Lan, M. G. Bellemare, and P. S. Castro, "Metrics and continuity in reinforcement learning," in *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 8261–8269.

[38] A. Touati, A. A. Taiga, and M. G. Bellemare, "Zooming for efficient model-free reinforcement learning in metric spaces," *arXiv preprint arXiv:2003.04069*, 2020.

[39] M. Pirotta, M. Restelli, and L. Bascetta, "Policy gradient in lipschitz markov decision processes," *Machine Learning*, vol. 100, pp. 255–283, 2015.

[40] X. Qi, J. Wang, Y. Chen, Y. Shi, and L. Zhang, "Lipsformer: Introducing lipschitz continuity to vision transformers," *arXiv preprint arXiv:2304.09856*, 2023.

[41] A. Virmaux and K. Scaman, "Lipschitz regularity of deep neural networks: analysis and efficient estimation," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[42] O. Sigaud and O. Buffet, *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

[43] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[44] J. Lally, M. Kazemi, and N. Paoletti, "Robust counterfactual inference in markov decision processes," *arXiv preprint arXiv:2502.13731*, 2025.

[45] S. Tsirtsis and M. Rodriguez, "Finding counterfactually optimal action sequences in continuous state spaces," *Advances in Neural Information Processing Systems*, vol. 36, pp. 3220–3247, 2023.

[46] B. Belousov, H. Abdulsamad, P. Klink, S. Parisi, and J. Peters, *Reinforcement learning algorithms: analysis and applications*. Springer, 2021.

[47] Z. Xiao, *Reinforcement learning: Theory and python implementation*. Springer, 2024.

[48] J. He, Z. Yang, J. Shi, C. Yang, K. Kim, B. Xu, X. Zhou, and D. Lo, "Curiosity-driven testing for sequential decision-making process," in *IEEE/ACM International Conference on Software Engineering*, 2024.

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

[50] X. Wan, T. Li, W. Lin, Y. Cai, and Z. Zheng, "Coverage-guided fuzzing for deep reinforcement learning systems," *Journal of Systems and Software*, vol. 210, p. 111963, 2024.

[51] H. Zhang, H. Chen, C. Xiao, B. Li, M. Liu, D. Boning, and C.-J. Hsieh, "Robust deep reinforcement learning against adversarial perturbations on state observations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 024–21 037, 2020.

[52] J. Kos and D. Song, "Delving into adversarial attacks on deep policies," *arXiv preprint arXiv:1705.06452*, 2017.

[53] Y. Lu, W. Sun, and M. Sun, "Towards mutation testing of reinforcement learning systems," *Journal of Systems Architecture*, vol. 131, p. 102701, 2022.

[54] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[55] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[56] J. Li, Z. Zheng, X. Du, H. Wang, and Y. Liu, "DRLMutation: A comprehensive framework for mutation testing in deep reinforcement learning systems," *ACM Transactions on Software Engineering and Methodology*, 2025, Just Accepted.

[57] D.-G. Thomas, M. Biagiola, N. Humbatova, M. Wardat, G. Jahangirova, H. Rajan, and P. Tonella, "μPRL: A mutation testing pipeline for deep reinforcement learning based on real faults," in *IEEE/ACM International Conference on Software Engineering*. IEEE Computer Society, 2025, pp. 2238–2250.

[58] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 67–78.

[59] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[60] P. Hu, Z. Zhang, W. K. Chan, and T. Tse, "An empirical comparison between direct and indirect test result checking approaches," in *International Workshop on Software Quality Assurance*, 2006, pp. 6–13.

[61] R. Florian, "Correct equations for the dynamics of the cart-pole system," *Center for Cognitive and Neural Studies*, vol. 63, 2007.

[62] I. Parberry, *Introduction to Game Physics with Box2D*. CRC Press, 2017.

[63] R. Li, H. Liu, P.-L. Poon, D. Towey, C.-A. Sun, Z. Zheng, Z. Q. Zhou, and T. Y. Chen, "Metamorphic relation generation: State of the art and research directions," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–25, 2025.

[64] J. Chen, C. Jia, Y. Yan, J. Ge, H. Zheng, and Y. Cheng, "A miss is as good as a mile: Metamorphic testing for deep learning operators," *ACM on Software Engineering*, vol. 1, no. FSE, pp. 2005–2027, 2024.

[65] S. Cho, S. Ruberto, and V. Terragni, "Metamorphic testing of large language models for natural language processing," in *IEEE International Conference on Software Maintenance and Evolution*, 2025, Just Accepted.