RICE: Breaking Through the Training Bottlenecks of Reinforcement Learning with Explanation

Zelei Cheng [*1] **Xian Wu** [*1] **Jiahao Yu** [1] **Sabrina Yang** [2] **Gang Wang** [3] **Xinyu Xing** [1]

Abstract

Deep reinforcement learning (DRL) is playing an increasingly important role in real-world applications. However, obtaining an optimally performing DRL agent for complex tasks, especially with sparse rewards, remains a significant challenge. The training of a DRL agent can be often trapped in a bottleneck without further progress. In this paper, we propose RICE, an innovative refining scheme for reinforcement learning that incorporates explanation methods to break through the training bottlenecks. The high-level idea of RICE is to construct a new initial state distribution that combines both the default initial states and critical states identified through explanation methods, thereby encouraging the agent to explore from the mixed initial states. Through careful design, we can theoretically guarantee that our refining scheme has a tighter sub-optimality bound. We evaluate RICE in various popular RL environments and real-world applications. The results demonstrate that RICE significantly outperforms existing refining schemes in enhancing agent performance.

1. Introduction

Deep reinforcement learning (DRL) has shown promising performance in various applications ranging from playing simulated games (Todorov et al., 2012; Mnih et al., 2013; Oh et al., 2016; Cai et al., 2023) to completing real-world tasks such as navigating autonomous vehicles and performing cybersecurity attacks and defenses (Bar-Zur et al., 2023; Vyas et al., 2023; Anderson et al., 2018; Wang et al., 2023). However, training an optimal DRL agent for complex tasks, particularly in environments with sparse rewards, presents a significant challenge. Often cases, the training of a DRL agent can hit a bottleneck without making further process: its sub-optimal performance becomes evident when it makes common mistakes or falls short of achieving the final goals.

When the DRL agent hits its training bottleneck, a refinement strategy can be considered, especially if the agent is already locally optimal. To refine the locally optimal DRL agent, one method is to analyze its behavior and patch the errors it made. A recent work (Cheng et al., 2023) proposes StateMask to identify critical states of the agent using an explanation method. One utility of StateMask is patching the agent's error, which fine-tunes the DRL agent starting from the identified critical states (denoted as "StateMask-R"). However, such an approach suffers from two drawbacks. On the one hand, initializing solely from critical states will hurt the diversity of initial states, which can cause overfitting (see Appendix D). On the other hand, fine-tuning alone cannot help the DRL agent jump out of the local optima. These observations drive us to rethink how to design a proper initial distribution and apply exploration-based techniques to patch previous errors.

Another reason behind the training bottleneck can be the poor choice of the training algorithm. Naturally, to improve performance, the developer needs to select another DRL training algorithm to re-train the DRL agent. However, for complex DRL tasks, re-training the agent *from scratch* is too costly. For instance, for AlphaStar (Vinyals et al., 2019) to attain grandmaster-level proficiency in StarCraft, its training period exceeds one month with TPUs. Retraining an agent of this level can incur a cost amounting to millions of dollars (Agarwal et al., 2022). Therefore, existing research has investigated the reuse of previous DRL training (as prior knowledge) to facilitate re-training (Ho & Ermon, 2016; Fu et al., 2018; Cai et al., 2022). The most recent example is Jump-Start Reinforcement Learning (JSRL) proposed by Uchendu et al. (2023) which leverages a pre-trained policy to design a curriculum to guide the training of a self-improving exploration policy. However, their selection of exploration frontiers in the curriculum is random, which cannot guarantee that the exploration frontiers have positive

[*]Equal contribution [1]Department of Computer Science, Northwestern University, Evanston, Illinois, USA [2]Presentation High School, San Jose, California, USA [3]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA. Correspondence to: Xinyu Xing <xinyu.xing@northwestern.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

returns. This motivates us to incorporate explanation methods to scrutinize the pre-trained policy and design more effective exploration frontiers.

In this work, we propose RICE[1], a **R**efining scheme for Re**I**nfor**C**ement learning with **E**xplanation. We first leverage a state-of-the-art explanation method to derive a step-level explanation for the pre-trained DRL policy. The explanation method identifies the most critical states (*i.e.,* steps that contribute the most to the final reward of a trajectory), which will be used to construct the exploration frontiers. Based on the explanation results, we construct a mixed initial state distribution that combines the default initial states and the identified critical states to prevent the overfitting problem. By forcing the agent to revisit these exploration frontiers, we further incentivize the agent to explore starting from the frontiers. Through exploration, the agent is able to expand state coverage, and therefore more effectively break through the bottlenecks of reinforcement learning training. Our theoretical analysis shows that this method achieves a tighter sub-optimality bound by utilizing this mixed initial distribution (see Section 3.4).

In addition, we introduce key improvements to the state-of-the-art explanation method StateMask (Cheng et al., 2023) to better facilitate our refining scheme. We reformulate the objective function and add a new reward bonus for encouraging blinding when training—this significantly simplifies the implementation without sacrificing the theoretical guarantee.

Evaluation and Findings. We evaluate the performance of RICE using four MuJoCo games and four DRL-based real-world applications, including cryptocurrency mining (Bar-Zur et al., 2023), autonomous cyber defense (Cage Challenge 2) (CAGE, 2022), autonomous driving (Li et al., 2022), and malware mutation (Raff et al., 2017). We show that the explanation derived from our new design demonstrates similar fidelity to the state-of-the-art technique StateMask (Cheng et al., 2023) with *significantly improved training efficiency*. With the explanation results, we show our refining method can produce higher performance improvements for the pre-trained DRL agent, in comparison with existing approaches including JSRL (Uchendu et al., 2023) and the original refining method from StateMask (Cheng et al., 2023).

In summary, our paper has the following contributions:

- We develop a refining strategy to break through the bottlenecks of reinforcement learning training with an explanation (which is backed up by a theoretical analysis). We show our refining method performs better than those informed by random explanation.
- We propose an alternative design of StateMask to explain the agent's policy in DRL-based applications. Experiments show that our explanation has comparable fidelity with StateMask while improving efficiency.
- With extensive evaluations and case studies, we illustrate the benefits of using RICE to improve a pre-trained policy.

[1]The source code of RICE can be found in https://github.com/chengzelei/RICE

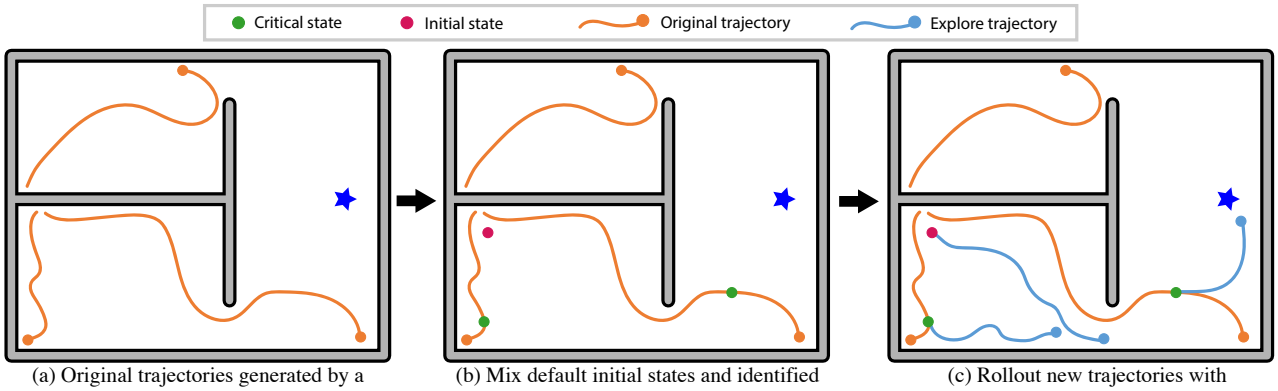
2. Related Work

2.1. Explanation-based Refining

Recently, there has been some work that leverages the DRL explanation to improve the agent's performance. These explanations can be derived from either human feedback or automated processes. Guan et al. (2021); Van Waveren et al. (2022) propose to utilize human feedback to correct the agent's failures. More specifically, when the agent fails, humans (can be non-experts) are involved to point out how to avoid such a failure (*i.e.,* what action should be done instead, and what action should be forbidden). Based on human feedback, the DRL agent gets refined by taking human-advised action in those important time steps and finally obtains the corrected policy. The downside is that it relies on humans to identify critical steps and craft rules for alternative actions. This can be challenging for a large action space, and the retraining process is ad-hoc and time-consuming. Cheng et al. (2023); Yu et al. (2023) propose to use step-level DRL explanation methods to automatically identify critical time steps and refine the agent accordingly. It initiates the refining process by resetting the environment to the critical states and subsequently resumes training the DRL agents from these critical states. Empirically, we observe that this refining strategy can easily lead to overfitting (see Appendix D). Instead, we propose a novel refining strategy with theoretical guarantees to improve the agent's performance.

2.2. Leveraging Existing Policy

The utilization of existing policies to initialize RL and enhance exploration has been explored in previous literature. Some studies propose to "roll-in" with an existing policy for better exploration, as demonstrated in works (Agarwal et al., 2020; Li et al., 2023). Similar to our approach, JSRL (Uchendu et al., 2023) incorporates a guide policy for roll-in, followed by a self-improving exploration policy. Technically, JSRL relies on a curriculum for the gradual update of the exploration frontier. However, the curriculum may not be able to truly reflect the key reasons why the guide policy succeeds or fails. Therefore, we propose to leverage the explanation method to automatically identify crucial states, facilitating the rollout of the policy by integrating these identified states with the default initial states. In Section 4, we



(a) Original trajectories generated by a pre-trained DRL policy.

(b) Mix default initial states and identified critical states based on explanation.

(c) Rollout new trajectories with RND-based exploration.

Figure 1. Given a pre-trained DRL policy that is not fully optimal (a), we propose the RICE algorithm that resets the RL agent to specific visited states (a mixture of default initial states and identified critical states) (b), followed by an exploration step initiated from these chosen states (c).

empirically demonstrate that JSRL performs poorly in our selected games. Chang et al. (2023) propose PPO++ that reset the environment to a mixture of the default initial states and the visited states of a guide policy (*i.e.,* a pre-trained policy). It can be viewed as a special case in our framework, *i.e.,* constructing a mixed initial distribution with a random explanation. However, we claim that not all visited states of a pre-trained policy are informative and our theoretical analysis and experiments both show that RICE based on our explanation method outperforms the refining method based on a random explanation.

3. Proposed Technique

3.1. Problem Setup and Assumption

We model the problem as a Markov Decision Process (MDP), which is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \rho, \mathcal{R}, \gamma \rangle$. In this tuple, $\mathcal{S}$ and $\mathcal{A}$ are the state and action set, where each s_t and a_t represents the state and action of the agent at time t. $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. $\gamma \in (0, 1)$ is the discount factor. For a policy $\pi(a|s)$: $\mathcal{S} \to \mathcal{A}$, the value function and Q-function is defined as $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s\right]$ and $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a\right]$. The advantage function for the policy π is denoted as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. We assume the initial state distribution is given by ρ: $s_0 \sim \rho$. The goal of RL is to find an optimal policy π^* that maximizes its expected total reward : $\pi^* = \arg\max_\pi \mathbb{E}_{s\sim\rho} [V^\pi(s)]$. Besides, we also introduce the state occupancy distribution and the state-action occupancy measure for π, denoted as $d^\pi_\rho(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \Pr^\pi (s_t = s \mid s_0 \sim \rho)$ and $d^\pi_\rho(s, a) = d^\pi_\rho(s)\pi(a|s)$.

In our setting, we have a pre-trained policy denoted as π, which may be sub-optimal. Our objective is to break through the training bottlenecks of the pre-trained policy with an explanation. Rather than re-training from scratch, we propose to utilize explanation to take full advantage of the guidance of the pre-trained policy π. Importantly, we do not assume knowledge of the original training algorithm used for policy π. And we make the following assumptions about the quality of π.

Assumption 3.1. Given a random policy π^r, we have $\mathbb{E}_{a\sim\pi^r}[A^\pi(s, a)] \leq 0, \forall s$.

Intuitively, the above assumption implies that taking an action based on a random policy π^r will provide a lower advantage than taking actions based on the policy π. This is a reasonable assumption since π is a pre-trained policy, thus it would perform much better than an untrained (*i.e.,* random) policy.

Assumption 3.2. The pre-trained policy π cover the states visited by the optimal policy π^*: $\left\| \frac{d^{\pi^*}_\rho}{d^\pi_\rho} \right\|_\infty \leq C$, where C is a constant.

In other words, Assumption 3.2 requires that the pre-trained policy visits all good states in the state space. Note that it is a standard assumption in the online policy gradient learning (Agarwal et al., 2021; Uchendu et al., 2023; Li et al., 2023) and is much weaker than the *single policy concentrateability coefficient* assumption (Rashidinejad et al., 2021; Xie et al., 2021), which requires the pre-trained policy visits all good state-action pairs. The ratio in Assumption 3.2 is also referred to as the distribution mismatch coefficient.

3.2. Technical Overview

Recall our goal is to refine the pre-trained DRL agent to break through the training bottlenecks. At a high level, the RICE algorithm integrates a roll-in step, where the RL agent is reset to specific visited states, followed by an exploration

step initiated from these chosen states. During the roll-in step, we draw inspiration from established RL-explanation methods (Puri et al., 2019; Guo et al., 2021; Cheng et al., 2023) to identify critical states, referred to as exploration frontiers, within the given policy π. As depicted in Figure 1, when presented with a trajectory sampled from the policy π, we employ a step-level explanation method – StateMask (Cheng et al., 2023) to identify the most crucial time steps influencing the final rewards in this trajectory. Subsequently, we guide the RL agent to revisit these selected states. The rationale behind revisiting these states lies in their ability to offer an expanded initial state distribution compared to ρ, thereby enabling the agent to explore diverse and relevant states it might otherwise neglect. Additionally, we introduce a mixing of these selected states with the initial states sampled from ρ. This mixing approach serves the purpose of preventing the agent from overfitting to specific states. In Section 3.4, we theoretically show that RICE achieves a tighter regret bound through the utilization of this mixed initial distribution.

Then, we propose an exploration-based method to further enhance the DRL agent's performance. The high-level idea is to incentivize the agent to explore when initiating actions from these frontiers. Intuitively, the pre-trained policy π might converge to a local optimal, as shown in Figure 1. Through exploration, we aim to expand state coverage by rewarding the agent for visiting novel states, thereby increasing the likelihood of successfully completing the task. Specifically, we utilize the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) for refining the DRL agent, leveraging the monotonicity of PPO.

3.3. Technique Detail

Step-level Explanation. We leverage a state-of-the-art explanation method StateMask (Cheng et al., 2023). At a high level, StateMask parameterizes the importance of the target agent's current time step as a neural network model (*i.e.,* mask network). This neural network takes the current state as input and then outputs this step's importance score with respect to the agent's final reward. To do so, StateMask learns a policy to "blind" the target agent at certain steps without changing the agent's final reward. Specifically, for an input state s_t, the mask net outputs a binary action a_t^m of either "zero" or "one", and the target agent will sample the action a_t from its policy. The final action is determined by the following equation

$$a_t \odot a_t^m = \begin{cases} a_t, & \text{if } a_t^m = 0\,, \\ a_{\text{random}} & \text{if } a_t^m = 1\,, \end{cases} \tag{1}$$

The mask net is then trained to minimize the following objective function:

$$J(\theta) = \min \left|\eta(\pi) - \eta(\bar{\pi})\right|, \tag{2}$$

Algorithm 1 Training the Mask Network.

Input: Target agent's policy π
Output: Mask network $\tilde{\pi}_\theta$
Initialization: Initialize the weights θ for the mask net $\tilde{\pi}_\theta$
$\theta_{old} \leftarrow \theta$
for iteration=1, 2, ... **do**
  Set the initial state $s_0 \sim \rho$
  $\mathcal{D} \leftarrow \emptyset$
  for t=0 to T **do**
    Sample $a_t \sim \pi(a_t|s_t)$
    Sample $a_t^m \sim \tilde{\pi}_{\theta_{old}}(a_t^m|s_t)$
    Compute the actual taken action $a \leftarrow a_t \odot a_t^m$
    $(s_{t+1}, R'_t) \leftarrow \texttt{env.step}(a)$ and record $(s_t, s_{t+1}, a_t^m, R'_t)$ in $\mathcal{D}$
  end for
  update $\theta_{old} \leftarrow \theta$ using $\mathcal{D}$ by PPO algorithm
end for

where π denotes the policy of the target agent (*i.e.,* our pre-trained policy), $\bar{\pi}$ denotes the policy of the perturbed agent (*i.e.,* integrating the random policy and the target agent π via the mask network $\tilde{\pi}$), $\eta(\cdot)$ is the expected total reward of an agent by following a certain policy. To solve the Eqn. (2) with monotonicaly guarantee, StateMask carefully designs a surrogate function and utilize the prime-dual methods to optimize the $\tilde{\pi}$. However, we can optimize the learning process of mask net within our setting to enhance simplicity. Specifically, we have the following theorem

Theorem 3.3. *Under Assumption 3.1, we have $\eta(\bar{\pi})$ upper-bounded by $\eta(\pi)$: $\eta(\bar{\pi}) \leq \eta(\pi)$.*

The proof of the theorem can be found in Appendix A. Leveraging this theorem, we can transform the objective function to $J(\theta) = \max \eta(\bar{\pi})$. With this reformulation, we can utilize the vanilla PPO algorithm to train the state mask without sacrificing the theoretical guarantee. However, naïvely maximizing the expected total reward may introduce a trivial solution to the problem which is to *not blind the target agent at all* (always outputs "0"). To tackle this problem, we add an additional reward by giving an extra bonus when the mask net outputs "1". The new reward can be written as $R'(s_t, a_t) = R(s_t, a_t) + \alpha a_t^m$ where α is a hyper-parameter. We present the learning process of the mask network in Algorithm 1. By applying this resolved mask to each state, we will be able to assess the state importance (*i.e.,* the probability of mask network outputting "0") at any time step.

Constructing Mixed Initial State Distribution. With the state mask $\tilde{\pi}$, we construct a mixed initial state distribution to expand the coverage of the state space. Initially, we randomly sample a trajectory by executing the pre-trained policy π. Subsequently, the state mask is applied to pinpoint the most important state within the episode τ by assessing the significance of each state. The resulting distribution of these identified critical states is denoted as $d_\rho^{\hat{\pi}}(s)$. Indeed,

Algorithm 2 Refining the DRL Agent.

Input: Pre-trained policy π, corresponding state mask $\tilde{\pi}$, default initial state distribution ρ, reset probability threshold p
Output: The agent's policy after refining π'
for iteration=1, 2, … **do**
  $\mathcal{D} \leftarrow \emptyset$
  RAND_NUM ← RAND(0,1)
  if RAND_NUM $< p$ **then**
    Run π to obtain a trajectory τ of length K
    Identify the most critical state s_t in τ via state mask $\tilde{\pi}$
    Set the initial state $s_0 \leftarrow s_t$
  else
    Set the initial state $s_0 \sim \rho$
  end if
  for t=0 to T **do**
    Sample $a_t \sim \pi(a_t|s_t)$
    $(s_{t+1}, R_t) \leftarrow \texttt{env.step}(a_t)$
    Calculate RND bonus $R_t^{RND} = \left\| f(s_{t+1}) - \hat{f}(s_{t+1}) \right\|^2$ with normalization
    Add $(s_t, s_{t+1}, a_t, R_t + \lambda R_t^{RND})$ to $\mathcal{D}$
  end for
  Optimize π_θ w.r.t PPO loss on $\mathcal{D}$
  Optimize $\hat{f}_\theta$ w.r.t. MSE loss on $\mathcal{D}$ using Adam
end for
$\pi' \leftarrow \pi_\theta$

in Section 3.4, we demonstrate that this re-weighting-like sampling is equivalent to sampling the state from a better policy $\hat{\pi}$. We then set the initial distribution μ as a mixture of the selected important states distribution $d_\rho^{\hat{\pi}}(s)$ and the original initial distribution of interest ρ: $\mu(s) = \beta d_\rho^{\hat{\pi}}(s) + (1-\beta)\rho(s)$, where β is a hyper-parameter.

Exploration with Random Network Distillation. Starting from the new initial state distribution, we continue training the DRL agent while encouraging the agent to do exploration. In contrast to goal-conditional RL (Ren et al., 2019; Ecoffet et al., 2019), which typically involve random exploration from identified frontiers, we advocate for the RL agent to explore novel states to increase the state coverage. Motivated by this, we adopt Random Network Distillation (RND) (Burda et al., 2018) which is proved to be an effective exploration bonus, especially in large and continuous state spaces where count-based bonuses (Bellemare et al., 2016; Ostrovski et al., 2017) can be hard to extend. Specifically, we directly utilize the PPO algorithm to update the policy π, except that we add the intrinsic reward to the task reward, that is, we optimize $R'(s_t, a_t) = R(s_t, a_t) + \lambda|f(s_{t+1}) - \hat{f}(s_{t+1})|^2$, where λ controls the trade-off between the task reward and exploration bonus. Along with the policy parameters, the RND predictor network $\hat{f}$ is updated to regress to the target network f. Note that, as the state coverage increases, RND bonuses decay to zero and a performed policy is recovered. We present our proposed refining method in Algorithm 2.

3.4. Theoretical Analysis

Finally, we provide theoretical analysis demonstrating that our refining algorithm can tighten the sub-optimality gap: SubOpt $:= V^{\pi^*}(\rho) - V^{\pi'}(\rho)$, (*i.e.,* the gap between the long-term reward collected by the optimal policy π^* and that obtained by the refined policy π' when starting from the default initial state distribution ρ).

In particular, we aim to answer the following two questions:
Q1:*What are the benefits of incorporating StateMask to determine the exploration frontier?*

Q2: *what advantages does starting exploration from the mixed initial distribution offer?*

To answer the questions, we first show that determining the exploration frontiers based on StateMask is equivalent to sampling states from a better policy compared to π. Then, we demonstrate that under the mixed initial distribution as introduced in Section 3.3, we could provide a tighter upper bound for the sub-optimality of trained policy π compared with randomly selecting visited states to form the initial distribution.

In order to answer **Q1**, we begin with Assumption 3.4 to assume the relationship between the policy value and the state distribution mismatch coefficient.

Assumption 3.4. For two polices π and $\hat{\pi}$, if $\eta(\hat{\pi}) \geq \eta(\pi)$, then we have $\left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\hat{\pi}}} \right\|_\infty \leq \left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\pi}} \right\|_\infty$.

Intuitively, this assumption posits that a superior policy would inherently possess a greater likelihood of visiting all favorable states. We give validation of this assumption in a 2-state MDP in Appendix B.1.

We further present Lemma 3.5 to answer **Q1**, *i.e.,* the benefits of incorporating StateMask to determine the exploration frontier. The proof of Lemma 3.5 can be found in Appendix B.2.

Lemma 3.5. *Given a pre-trained policy π, our MaskNet-based sampling approach in Section 3.3 is equivalent to sampling states from a state occupation distribution induced by an improved policy $\hat{\pi}$.*

In order to answer **Q2**, we start with presenting Theorem 3.6 to bound the sub-optimality via the state distribution mismatch coefficient.

Theorem 3.6. *Assume that for the refined policy π', $\mathbb{E}_{s\sim d_\mu^{\pi'}}\left[\max_a A^{\pi'}(s,a)\right] < \epsilon$. For two initial state distributions μ and ρ, we have the following bound (Kakade & Langford, 2002)*

$$V^{\pi^*}(\rho) - V^{\pi'}(\rho) \leq \mathcal{O}\left(\frac{\epsilon}{(1-\gamma)^2} \left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\hat{\pi}}} \right\|_\infty\right). \quad (3)$$

The proof of Theorem 3.6 can be found in Appendix B.3. It indicates that the upper bound on the difference between the performance of the optimal policy π^* and that of the policy π' after refining is proportional to $\left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\tilde{\pi}}} \right\|_\infty$. With Assumption 3.4 and Lemma 3.5, we now claim that our refining method with our explanation could further tighten the sub-optimality gap via reducing the distribution mismatch coefficient compared with forming an initial distribution by random selecting visited states, *i.e.,* with a random explanation.

Claim 1. *We can form a better (mixed) initial state distribution μ with our explanation method and tighten the upper bound of $V^{\pi^*}(\rho) - V^{\pi'}(\rho)$ compared with random explanation.*

The details of the analysis can be found in Appendix B.4. Based on Assumption 3.2 and Claim 1, we can learn to perform as well as the optimal policy as long as the visited states of the optimal policy are covered by the (mixed) initial distribution.

4. Evaluation

In this section, we start with our experiment setup and design, followed by experiment results and analysis. We provide additional evaluation details in Appendix C.

4.1. Experiment Setup

Environment Selection. We select eight representative environments to demonstrate the effectiveness of `RICE` across two categories: simulated games (Hopper, Walker2d, Reacher, and HalfCheetah of the MuJoCo games) and real-world applications (selfish mining, network defense, autonomous driving, and malware mutation) [2]. We additionally run the experiments in the three sparse MuJoCo games introduced by Mazoure et al. (2019). The details of these applications can be found in Appendix C.2.

Baseline Explanation Methods. Since our explanation method proposes an alternative design of StateMask, the first baseline is StateMask. We compare our explanation method with StateMask to show the equivalence and efficiency of our method. Additionally, we introduce "Random" as a baseline explanation method. "Random" identifies critical steps by randomly selecting a visited state as the critical state.

Baseline Refining Methods. We compare our refining method with three baselines. The first baseline is "PPO fine-tuning" (Schulman et al., 2017), *i.e.,* lowering the learning rate and continuing training with the PPO algorithm. The second baseline is a refining method introduced by StateMask (Cheng et al., 2023), *i.e.,* resetting to the critical state and continuing training from the critical state. The third baseline is Jump-Start Reinforcement Learning (referred to as "JSRL") (Uchendu et al., 2023). JSRL introduces a guided policy π_g to set up a curriculum to train an exploration policy π_e. Through initializing $\pi_e = \pi_g$, we can transform JSRL to be a refining method that can further improve the performance of the guided policy.

Evaluation Metrics. To evaluate the fidelity of the generated explanation, we utilize an established fidelity score metric defined in StateMask (Cheng et al., 2023). The idea is to use a sliding window to step through all time steps and then choose the window with the highest average importance score (scored by the explanation method). The width of the sliding window is l while the whole trajectory length is L. Then we randomize the action(s) at the selected critical step(s) in the selected window (*i.e.,* masking) and measure the average reward change as d. Additionally, we denote the maximum possible reward change as d_{max}. Therefore, the fidelity score is calculated as $\log(d/d_{max}) - \log(l/L)$. A higher fidelity score indicates higher fidelity.

For the applications with dense rewards except the malware mutation application, we measure the reward of the target agent before and after refining. In the case of the malware mutation application, we report the "final reward" as the probability of evading the malware detector, both before and after refining. For the applications with sparse rewards, we report the performance during the refining process.

4.2. Experiment Design

We use the following experiments to evaluate the fidelity and efficiency of the explanation method, the effectiveness of the refining method and other factors that influenced the system performance (*e.g.,* alternative design choices, hyper-parameters).

Experiment I. To show the equivalence of our explanation method with StateMask, we compare the fidelity of our method with StateMask. Given a trajectory, the explanation method first identifies and ranks top-K important time steps. An accurate explanation means the important time steps have significant contributions to the final reward. To validate this, we let the agent fast-forward to the critical step and force the target agent to take random actions. Then we follow the target agent's policy to complete the rest of the time steps. If the explanation is accurate, we expect a major change to the final reward by randomizing the actions at the important steps. We compute the fidelity score of each explanation method as mentioned in StateMask across 500 trajectories. We set $K = 10\%, 20\%, 30\%, 40\%$ and report

[2] These are representative security applications that have a significant impact on the security community (Anderson et al., 2018) and they represent RL tasks with sparse rewards, which are common in security applications.

the fidelity of the selected methods under each setup. We repeat each experiment 3 times with various random seeds and report the mean and standard deviation. Additionally, to show the efficiency of our design, we report the training time of the mask network using StateMask and our method when given a fixed number of training samples.

Experiment II. To show the effectiveness of the refining method, we compare the agent's performance after refining using our method and three aforementioned baseline methods, *i.e.,* PPO fine-tuning (Schulman et al., 2017), StateMask's fine-tuning from critical steps (Cheng et al., 2023), and Jump-Start Reinforcement Learning (Uchendu et al., 2023). For this experiment, all the refining methods use the *same explanation* generated by our explanation method if needed, to ensure a fair comparison. Additionally, we conduct a qualitative study to understand how our refining method influences agent behavior and performance.

Experiment III To investigate how the quality of explanation affects the downstream refining process, we run our proposed refining method based on the critical steps identified by different explanation methods (Random, StateMask, and our method) and compare the agent's performance after refining.

Experiment IV. To show the versatility of our method, we examine the refining performance when the pre-trained agent was trained by other algorithms such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018). First, we obtain a pre-trained SAC agent and then use Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016) to learn an approximated policy network. We compare the refining performance using our method against baseline methods, *i.e.,* PPO fine-tuning (Schulman et al., 2017), StateMask's fine-tuning from critical steps (Cheng et al., 2023), and Jump-Start Reinforcement Learning (Uchendu et al., 2023). In addition, we also include fine-tuning the pre-trained SAC agent with the SAC algorithm as a baseline.

Experiment V. We test the impact of hyper-parameter choices for two primary hyper-parameters for refining: p (used to control the mixed initial state distribution) and λ (used to control the exploration bonus). For our refining method, we vary p from $\{0, 0.25, 0.5, 0.75, 1\}$ and vary λ from $\{0, 0.1, 0.01, 0.001\}$. By examining the agent's performance with various λ values, we can further investigate the necessity of the exploration bonus. Additionally, we evaluate the choice of α for our explanation method (used to control the mask ratio for the mask network). Specifically, we vary α from $\{0.01, 0.001, 0.0001\}$.

4.3. Experiment Results

Fidelity and Efficiency of Explanation. We compare the fidelity scores of our method with StateMask in all applications and provide the full results in Figure 5 of Appendix C.3. We observe that the fidelity scores of StateMask and our method are comparable. Furthermore, We evaluate the efficiency of our explanation method compared with StateMask. We report the cost time and the number of samples when training our explanation method and StateMask in Table 4 of Appendix C.3. We observe an average of 16.8% drop in the training time compared with StateMask. The reason is that the training algorithm of the mask network in StateMask involves an estimation of the discounted accumulated reward with respect to the current policy of the perturbed agent and the policy of the target agent which requires additional computation cost. In contrast, our design only adds an additional term to the reward which is simple but effective.

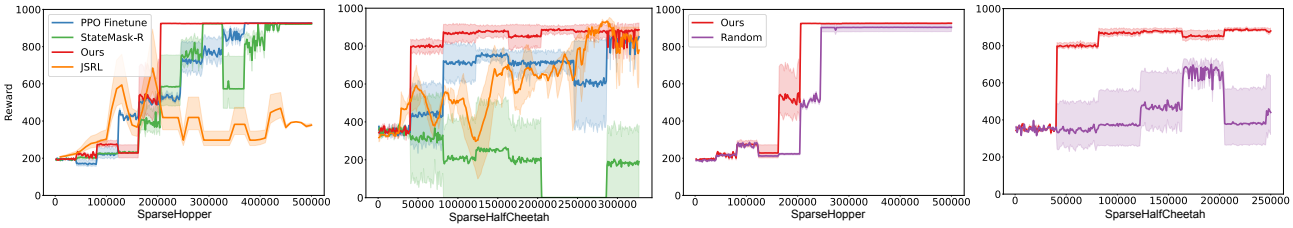
Effectiveness of Refining. We compare the agent's performance after refining using different retaining methods across all applications with dense rewards in Table 1. The performance is measured by the final reward of the refined agent. In most applications, rewards are typically assigned positive values. However, in Cage Challenge 2, the reward is designed to incorporate negative values (see Appendix C.2). We have three main observations. First, we observe that our refining method can bring the largest improvement for the target agent in all applications. Second, we find that the PPO fine-tuning method only has marginal improvements for the agents due to its incapability of jumping out of local optima. Third, the refining method proposed in StateMask (which is to start fine-tuning only from critical steps) cannot always improve the agent's performance. The reason is that this refining strategy can cause overfitting and thus harm the agent's performance. We illustrate this problem in greater detail in a case study of Malware Mutation in Appendix D. It is also worth mentioning that we discover design flows of Malware Mutation and present the details in Appendix D.

We also run our experiments of varying refining methods on selected MuJoCo games with sparse rewards. Figure 2 shows the results of our method against other baselines in SparseHopper and SparseHalfCheetah games. We observe that our refining method has significant advantages over other baselines with respect to final performance and refining efficiency. Through varying explanation methods, we confirm that the contribution should owe to our explanation method. We leave the refining results of the SparseWalker2d game and the hyper-parameter sensitivity results of all sparse MuJoCo games in Appendix C.4.

In addition to numerical results, we also provide a qualitative analysis of the autonomous driving case to understand how `RICE` influences agent behavior and performance, particularly in a critical state, in Appendix C.5. We visualize the agent's behavior before and after refining the agent to show that `RICE` is able to help the agent break through

Table 1. **Agent Refining Performance**—"No Refine" indicates the target agent's performance before refining. For the first group of experiments (left), we fixed the explanation method to our method (mask network) and varied the refining methods. For the second group of experiments (right), we fixed the refining method to our method and varied the explanation methods. We report the mean value (standard deviations) of the final reward after refining. A higher value is better.

Task	No Refine	Fix Explanation; Vary Refine Methods				Fix Refine; Vary Explanation Methods		
		PPO	JSRL	StateMask-R	**Ours**	Random	StateMask	**Ours**
Hopper	3559.44 (19.15)	3638.75 (16.67)	3635.08 (9.82)	3652.06 (8.63)	**3663.91 (20.98)**	3648.98 (39.06)	3661.86 (19.95)	**3663.91 (20.98)**
Walker2d	3768.79 (18.68)	3965.63 (9.46)	3963.57 (6.73)	3966.96 (3.39)	**3982.79 (3.15)**	3969.64(6.38)	3982.67 (5.55)	**3982.79 (3.15)**
Reacher	-5.79 (0.73)	-3.04 (0.04)	-3.23 (0.26)	-3.45 (0.32)	**-2.66 (0.03)**	-3.11 (0.42)	-2.69 (0.28)	**-2.66 (0.03)**
HalfCheetah	2024.09 (28.34)	2133.31 (4.11)	2128.04 (0.91)	2085.28 (1.92)	**2138.89 (3.22)**	2132.01 (0.76)	2136.23 (0.49)	**2138.89 (3.22)**
Selfish Mining	14.36 (0.24)	14.93 (0.45)	14.88 (0.51)	14.53 (0.33)	**16.56 (0.63)**	15.09 (0.28)	16.49 (0.46)	**16.56 (0.63)**
Cage Challenge 2	-23.64 (0.27)	-23.58 (0.37)	-22.97 (0.57)	-26.98 (0.84)	**-20.02 (0.32)**	-25.94 (2.34)	-20.07 (1.33)	**-20.02 (0.32)**
Auto Driving	10.30 (2.25)	13.37 (3.10)	11.26 (3.66)	7.62 (1.77)	**17.03 (1.65)**	11.72 (1.77)	16.28 (2.33)	**17.03 (1.65)**
Malware Mutation	42.20 (6.86)	49.33 (8.59)	43.10 (7.24)	50.13 (8.14)	**57.53 (8.71)**	48.60 (7.60)	57.16 (8.51)	**57.53 (8.71)**



(a) Fix Explanation; Vary Refine (b) Fix Refine; Vary Explanation

Figure 2. **Agent Refining Performance in two Sparse MuJoCo Games**—For Group (a), we fix the explanation method to our method (mask network) if needed while varying refining methods. For Group (b), we fix the refining method to our method while varying the explanation methods.

the bottleneck based on the identified critical states of the failure.

Refining based on Different Explanations. To examine how the quality of explanation affects the downstream refining process, we present Table 1. We run our proposed refining method based on the critical steps identified by ours and Random. We have two main observations. First, using the explanation generated by our mask network, the refining achieves the best outcome across all applications. Second, using the explanation generated by our explanation significantly outperforms the random baseline. This aligns with our theoretical analysis that our refining framework provides a tighter bound for the sub-optimality.

Refining a Pre-trained Agent of Other Algorithms. To show that our framework is general to refine pre-trained agents that were not trained by PPO algorithms, we do experiments on refining a SAC agent in the Hopper game. Figure 3 demonstrates the advantage of our refining method against other baselines when refining a SAC agent. Additionally, we observe that fine-tuning the DRL agent with the SAC algorithm still suffers from the training bottleneck while switching to the PPO algorithm provides an opportunity to break through the bottleneck. We provide the refining curves when varying hyper-parameters p and λ in Appendix C.3.

Impact of Hyper-parameters. Due to space limit, we provide the sensitivity of hyper-parameters p, λ, and α in

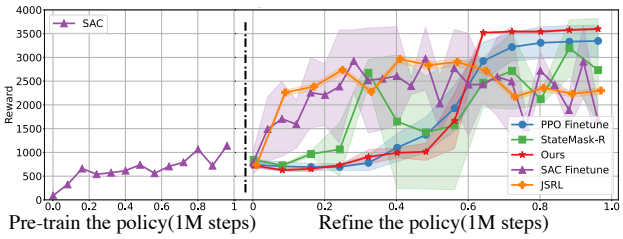


Figure 3. **SAC Agent Refining Performance in Hopper Game**—In the left part, we show the training curve of obtaining a pre-trained policy through the SAC algorithm. In the right part, we show the refining curves of different methods.

Appendix C.3. We have three main observations.

First, p controls the mixing ratio of critical states (identified by the explanation method) and the initial state distribution for refining. The performance is low when $p = 0$ (all starting from the default initial distribution) or $p = 1$ (all starting from the identified critical states). The performance has significant improvements when $0 < p < 1$, *i.e.,* using a mixed initial state distribution. Across all applications, we observe that setting p to 0.25 or 0.5 is most beneficial. A mixed initial distribution can help eliminate the problem of overfitting.

Second, as long as $\lambda > 0$ (thereby enabling exploration), there is a noticeable improvement in performance, highlighting the importance of exploration in refining the pre-trained agent. The result is less sensitive to the specific value of

λ. In general, a λ value of 0.01 yields good performance across all four applications.

Third, recall that the hyper-parameter α is to control the bonus of blinding the target agent when training the mask network. We vary α from $\{0.01, 0.001, 0.0001\}$ and find that our explanation method is not that sensitive to α.

5. Discussion

Applicability. RICE is suitable for DRL applications that are trained within *controllable environment* (*e.g.,* simulators), in order to generate explanations. In fact, most of today's DRL applications rely on some form of simulator for their training. For example, for safety-critical applications such as autonomous driving, DRL agents are usually designed, trained, and tested in a simulated environment first before moving them to real-world testing. Simulation platforms broadly include Carla (Dosovitskiy et al., 2017) and MetaDrive (Li et al., 2022) which have been used to facilitate the training of DRL agents (Zhang et al., 2021; Wang et al., 2023; Peng et al., 2022). Therefore, RICE should be applicable to such DRL systems (especially during their development phase) for refining a pre-trained DRL agent.

Warm Start vs Cold Start. As is mentioned in Section 3, our method requires a "warm start" setting, *i.e.,* the agent has good coverage of the state distribution of the optimal policy. Even if the agent has good coverage of the state distribution, it does not necessarily mean that the agent has already learned a good policy due to the potential of choosing wrong actions (Uchendu et al., 2023). Therefore, the training bottleneck can still exist under a good coverage of the state distribution. In contrast, Our method does not work well in a "cold start" setting, *i.e.,* when the state coverage of the pre-trained policy is extremely poor. In that case, step-level explanation methods cannot give useful help and our method is actually equivalent to the RND method [3].

Critical State Filtering. Though RICE identifies critical states based on their necessity for achieving good outcomes, it does not fully consider their importance for further agent learning. For instance, a state might be deemed critical, yet the trained agent could have already converged to the optimal action for that state. In such cases, resetting the environment to this state doesn't significantly benefit the learning process. Future work could explore additional filtering of critical states using metrics such as policy convergence or temporal difference (TD) errors, which may help concentrate efforts and accelerate refinement.

[3]We provide an example of Mountain Car game in Appendix E to illustrate this limitation.

6. Conclusion

In this paper, we present RICE to break through bottlenecks of reinforcement learning training with explanation. We propose an alternative design of StateMask to provide high-fidelity explanations for DRL agents' behaviors, by identifying critical time steps that contribute the most to the agent's success/failure. We encourage the agent to explore starting from a mixture of default initial states and the identified critical states. Compared with existing refining strategies, we empirically show that our method brings the largest improvement after refining with theoretical guarantees.

Acknowledgements

This project was supported in part by Northwestern University TGS Fellowship and NSF Grants 2225234, 2225225, 2229876, 1955719, and 2055233.

Impact Statement

This paper presents work whose goal is to advance the field of reinforcement learning with explanation. There are many potential social impacts of our work. Our approach provides a feasible solution to break through the training bottlenecks of reinforcement learning with explanation, which is an automatic process and saves manual effort.

However, it is also worth noting the potential negative societal impacts of our work. Some of the real-world applications we select such as malware mutation can create attack examples that may bring additional ethical concerns. In the realm of security research, the ultimate goal of these tasks is to generate stronger testing cases to enhance the defense, and it is standard practice. Take malware mutation as an example, the produced samples can be used to proactively improve the robustness and effectiveness of malware detection systems (*e.g.,* through adversarial training), thereby benefiting cybersecurity defense (Yang et al., 2017).

References

GitHub - bfilar/malware_rl: Malware Bypass Research using Reinforcement Learning — github.com. https://github.com/bfilar/malware_rl, a.

GitHub - cage-challenge/cage-challenge-2: TTCP CAGE Challenge 2 — github.com. https://github.com/cage-challenge/cage-challenge-2, b.

GitHub - john-cardiff/-cyborg-cage-2 — github.com. https://github.com/john-cardiff/-cyborg-cage-2, c.

GitHub - roibarzur/pto-selfish-mining: Code repository for technical papers about selfish mining analysis. — github.com. https://github.com/roibarzur/pto-selfish-mining, d.

Mountain car continuous. https://mgoulao.github.io/gym-docs/environments/classic_control/mountain_car_continuous/. Accessed: 2024-05-24.

Agarwal, A., Henaff, M., Kakade, S., and Sun, W. Pc-pg: Policy cover directed exploration for provable policy gradient learning. *Proc. of NeurIPS*, 2020.

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 2021.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. In *Proc. of NeurIPS*, 2022.

Anderson, H. S., Kharkar, A., Filar, B., Evans, D., and Roth, P. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

Bar-Zur, R., Abu-Hanna, A., Eyal, I., and Tamar, A. Werlman: To tackle whale (transactions), go deep (rl). In *Proc. of IEEE S&P*, 2023.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Proc. of NeurIPS*, 2016.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *Proc. of ICLR*, 2018.

CAGE. Ttcp cage challenge 2. In *Proc. of AAAI-22 Workshop on Artificial Intelligence for Cyber Security (AICS)*, 2022.

Cai, X.-Q., Ding, Y.-X., Chen, Z., Jiang, Y., Sugiyama, M., and Zhou, Z.-H. Seeing differently, acting similarly: Heterogeneously observable imitation learning. In *Proc. of ICLR*, 2022.

Cai, X.-Q., Zhang, Y.-J., Chiang, C.-K., and Sugiyama, M. Imitation learning from vague feedback. In *Proc. of NeurIPS*, 2023.

Chang, J. D., Brantley, K., Ramamurthy, R., Misra, D., and Sun, W. Learning to generate better than your llm. *arXiv preprint arXiv:2306.11816*, 2023.

Cheng, Z., Wu, X., Yu, J., Sun, W., Guo, W., and Xing, X. Statemask: Explaining deep reinforcement learning through state mask. In *Proc. of NeurIPS*, 2023.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. CARLA: An open urban driving simulator. In *Proc. of CoRL*, pp. 1–16, 2017.

drive Contributors, D. DI-drive: OpenDILab decision intelligence platform for autonomous driving simulation. https://github.com/opendilab/DI-drive, 2021.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 2021.

Erez, T., Tassa, Y., and Todorov, E. Infinite-horizon model predictive control for periodic tasks with contacts. *Robotics: Science and systems VII*, pp. 73, 2012.

Eyal, I. and Sirer, E. G. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61 (7):95–102, 2018.

Eysenbach, B., Salakhutdinov, R., and Levine, S. The information geometry of unsupervised reinforcement learning. In *Proc. of ICLR*, 2021.

Fu, J., Luo, K., and Levine, S. Learning robust rewards with adverserial inverse reinforcement learning. In *Proc. of ICLR*, 2018.

Guan, L., Verma, M., Guo, S. S., Zhang, R., and Kambhampati, S. Widening the pipeline in human-guided reinforcement learning with explanation and context-aware data augmentation. In *Proc.of NeurIPS*, 2021.

Guo, W., Wu, X., Khan, U., and Xing, X. Edge: Explaining deep reinforcement learning policies. In *Proc. of NeurIPS*, 2021.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. of ICML*, pp. 1861–1870, 2018.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Proc. of NeurIPS*, 2016.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *Proc. of ICML*, 2002.

Li, Q., Peng, Z., Feng, L., Zhang, Q., Xue, Z., and Zhou, B. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Li, Q., Zhai, Y., Ma, Y., and Levine, S. Understanding the complexity gains of single-task rl with a curriculum. In *Proc. of ICML*, pp. 20412–20451, 2023.

Mazoure, B., Doan, T., Durand, A., Hjelm, R. D., and Pineau, J. Leveraging exploration in off-policy algorithms via normalizing flows. In *Proc. of CoRL*, 2019.

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

Oh, J., Chockalingam, V., Lee, H., et al. Control of memory, active perception, and action in minecraft. In *Proc. of ICML*, 2016.

Oh, J., Guo, Y., Singh, S., and Lee, H. Self-imitation learning. In *Proc. of ICML*, pp. 3878–3887, 2018.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. Count-based exploration with neural density models. In *Proc. of ICML*, 2017.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Proc. of NeurIPS*, 2019.

Peng, Z., Li, Q., Liu, C., and Zhou, B. Safe driving via expert guided policy optimization. In *Proc. of CoRL*, 2022.

Puri, N., Verma, S., Gupta, P., Kayastha, D., Deshmukh, S., Krishnamurthy, B., and Singh, S. Explain your move: Understanding agent actions using specific and relevant feature attribution. In *Proc. of ICLR*, 2019.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435*, 2017.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

Rashidinejad, P., Zhu, B., Ma, C., Jiao, J., and Russell, S. Bridging offline reinforcement learning and imitation learning: A tale of pessimism. In *Proc. of NeurIPS*, 2021.

Ren, Z., Dong, K., Zhou, Y., Liu, Q., and Peng, J. Exploration via hindsight goal generation. In *Proc. of NeurIPS*, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. In *Proc. of ICML*, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Proc. of IROS*, 2012.

Uchendu, I., Xiao, T., Lu, Y., Zhu, B., Yan, M., Simon, J., Bennice, M., Fu, C., Ma, C., Jiao, J., et al. Jump-start reinforcement learning. In *Proc. of ICML*, 2023.

Van Waveren, S., Pek, C., Tumova, J., and Leite, I. Correct me if i'm wrong: Using non-experts to repair reinforcement learning policies. In *Proc. of HRI*, 2022.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019.

Vyas, S., Hannay, J., Bolton, A., and Burnap, P. P. Automated cyber defence: A review. *arXiv preprint arXiv:2303.04926*, 2023.

Wang, X., Zhang, J., Hou, D., and Cheng, Y. Autonomous driving based on approximate safe action. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, Y., Su, H., and Zhu, J. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 2022.

Xie, T., Jiang, N., Wang, H., Xiong, C., and Bai, Y. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. In *Proc. of NeurIPS*, 2021.

Yang, W., Kong, D., Xie, T., and Gunter, C. A. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proc. of ACSAC*, 2017.

Yu, J., Guo, W., Qin, Q., Wang, G., Wang, T., and Xing, X. Airs: Explanation for deep reinforcement learning based security applications. In *Proc. of USENIX Security*, 2023.

Zhang, Z., Liniger, A., Dai, D., Yu, F., and Van Gool, L. End-to-end urban driving by imitating a reinforcement learning coach. In *Proc. of ICCV*, 2021.

A. Proof of Theorem 3.3

Based on the Performance Difference Lemma (Kakade & Langford, 2002), we have

$$\eta(\bar{\pi}) - \eta(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}\mathbb{E}_{a\sim\bar{\pi}(\cdot|s)}A^\pi(s,a), \tag{4}$$

where π is the policy of the target agent, $\bar{\pi}$ is the perturbed policy, ρ is the initial distribution, and γ is the discount rate.

Note that the perturbed policy $\bar{\pi}$ is a mixture of the target agent's policy π and a random policy π^r (*i.e.,* $\bar{\pi}(\cdot|s) = \tilde{\pi}(a^e = 0|s)\pi(\cdot|s) + \tilde{\pi}(a^e = 1|s)\pi^r(\cdot|s)$). Denote the probability of the mask network outputting 0 at state s as $\tilde{\pi}(a^e = 0|s) = \xi(s)$ and the probability of the mask network outputting 1 at state s as $\tilde{\pi}(a^e = 1|s) = 1 - \xi(s)$ Given the fact that $A^\pi(s, \pi(\cdot|s)) = \mathbb{E}_{a\sim\pi(s)}A^\pi(s,a) = 0$, we have

$$\begin{aligned}
\eta(\bar{\pi}) - \eta(\pi) &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}\mathbb{E}_{a\sim\bar{\pi}(\cdot|s)}A^\pi(s,a) \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}\sum_a \bar{\pi}(a|s)A^\pi(s,a) \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}\sum_a \xi(s)\pi(a|s)A^\pi(s,a) + \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}\sum_a (1-\xi(s))\pi^r(a|s)A^\pi(s,a) \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}\xi(s)\mathbb{E}_{a\sim\pi(\cdot|s)}A^\pi(s,a) + \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}(1-\xi(s))\mathbb{E}_{a\sim\pi^r(\cdot|s)}A^\pi(s,a) \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\rho^{\bar{\pi}}}(1-\xi(s))\mathbb{E}_{a\sim\pi^r(\cdot|s)}A^\pi(s,a) \leq 0.
\end{aligned} \tag{5}$$

Therefore, we show that $\eta(\bar{\pi})$ is upper bounded by $\eta(\pi)$ given Assumption 3.1.

B. Theoretical Guarantee

B.1. Validation of Assumption 3.4 in a 2-state MDP

In a 2-state MDP, we have two different states, namely, s_A and s_B. The state distribution of any policy π follows $d_\rho^\pi(s_A) + d_\rho^\pi(s_B) = 1$. As such, the set of feasible state marginal distribution can be described by a line $[(0,1),(1,0)]$ in $\mathbb{R}^2$. Let's denote vector $\mathbf{s} = [s_A, s_B]$. The expected total reward of a policy π can be represented as $\eta(\pi) =< d_\rho^\pi(\mathbf{s}), R(\mathbf{s}) >$ (Eysenbach et al., 2021), where $R(\mathbf{s}) = [R(s_A), R(s_B)]$. Figure 4 shows the area of achievable state distribution via the initial state distribution ρ (highlighted in orange).

It should be noted that not all the points in the line $[(0,1),(1,0)]$ corresponded to a valid Markovian policy. However, for any convex combination of valid state occupancy measures, there exists a Markovian policy that has this state occupancy measure. As such, the policy search occurs within a convex polytope, essentially a segment (*i.e.,* , marked in orange) along this line. In Figure 4, we visualize $R(\mathbf{s})$ as vectors starting at the origin. Since $V^{\hat{\pi}}(\rho) \geq V^\pi(\rho)$, We mark $d_\rho^{\hat{\pi}}(\mathbf{s})$ closer to $R(\mathbf{s})$ (*i.e.,* the inner product between $d_\rho^{\hat{\pi}}(\mathbf{s})$ and $R(\mathbf{s})$ and is larger than $d_\rho^\pi(\mathbf{s})$ and $R(\mathbf{s})$). The following theorem explains how we determine the location of the location of $d_\rho^{\pi^*}(\mathbf{s})$ in Figure 4.

Theorem B.1 (Fact 1 (Eysenbach et al., 2021))**.** *For every state-dependent reward function, among the set of policies that maximize that reward function is one that lies at a vertex of the state marginal polytope.*

According to Theorem B.1, $d_\rho^{\pi^*}(\mathbf{s})$ located at either vertex in the orange segment. Since π^* is the optimal policy, it lies at the vertex that has the larger inner product within $R(s)$. Once the position of $d_\rho^{\pi^*}(\mathbf{s})$ is determined, we can easily find $\left\|\frac{d_\rho^{\pi^*}(s)}{d_\rho^{\hat{\pi}}(s)}\right\|_\infty \leq \left\|\frac{d_\rho^{\pi^*}(s)}{d_\rho^{\pi}(s)}\right\|_\infty$ based on Figure 4.

B.2. Proof of Lemma 3.5

Proof. Since our explanation method provides the importance of each state, we could view the sampling based on the state's importance as a reweighting of the state occupancy measure. Mathematically, it can be expressed as $d_\rho^{\hat{\pi}}(s) = d_\rho^\pi(s)w(s)$,

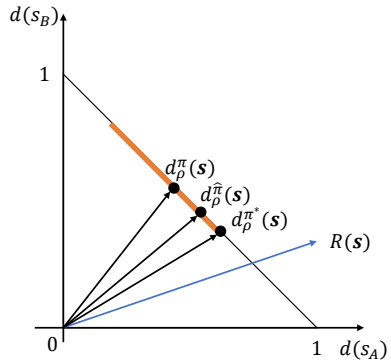


Figure 4. Visualization of state occupancy measures with respect to different policies and the reward function in a 2-state MDP.

where $\hat{\pi}$ is the equivalent policy of reweighting the original policy π and $w(s)$ is the weight provided by the mask network. Although the mask network takes the current input state as input, it implicitly considers the current action as well, as detailed by StateMask (Cheng et al., 2023). Consequently, a more accurate formulation is $d_\rho^{\hat{\pi}}(s,a) = d_\rho^{\pi}(s,a)w(s,a)$, where $w(s,a)$ represent the weight assigned by mask network.

Recall that our proposed explanation method is to randomize actions at non-critical steps, which essentially considers the value of $Q_{diff} = Q^\pi(s,a) - \mathbb{E}_{a' \in \mathcal{A}}[Q^\pi(s,a')]$. In fact, a larger Q_{diff} indicates current time step is more critical to the agent's final reward. Our mask network approximates the value of Q_{diff} via the deep neural network to determine the importance of each step, which implies $w(s,a) \propto Q_{diff} \propto Q^\pi(s,a)$.

Next, we aim to prove that our MaskNet-based sampling approach is equivalent to sampling from a better policy $\hat{\pi}$.

First, the equivalent policy $\hat{\pi}$ after reweighting can be expressed as

$$\hat{\pi}(a|s) = \frac{d_\rho^{\hat{\pi}}(s,a)}{d_\rho^{\hat{\pi}}(s)} = \frac{d_\rho^{\pi}(s,a)w(s,a)}{d_\rho^{\hat{\pi}}(s)} = w(s,a)\pi(a|s)\frac{d_\rho^{\pi}(s)}{d_\rho^{\hat{\pi}}(s)}. \tag{6}$$

Further , we would like to show that if $w(s,a) = f(Q^\pi(s,a))$ where $f(\cdot)$ is a monotonic increasing function, $\hat{\pi}$ is uniformly as good as,or better than π, *i.e.,* $V^{\hat{\pi}}(s) \geq V^\pi(s)$.

Proposition B.2. *Suppose two policies $\hat{\pi}$ and π satisfy $g\left(\hat{\pi}(a|s)\right) = g(\pi(a|s)) + h\left(s, Q^\pi(s,a)\right)$ where $g(\cdot)$ is a monotonically increasing function, and $h(s,\cdot)$ is monotonically increasing for any fixed s . Then we have $V^{\hat{\pi}}(s) \geq V^\pi(s)$, $\forall s \in \mathcal{S}$.*

Proof. For a given s, we partition the action set $\mathcal{A}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$.

$$\mathcal{A}_1 \triangleq \{a \in \mathcal{A} | \hat{\pi}(a|s) \geqslant \pi(a|s)\}.$$

$$\mathcal{A}_2 \triangleq \{a \in \mathcal{A} | \hat{\pi}(a|s) < \pi(a|s)\}.$$

Thus, $\forall a_1 \in \mathcal{A}_1, \forall a_2 \in \mathcal{A}_2$, we have

$$\begin{aligned} h\left(s, Q^\pi(s,a_1)\right) &= g\left(\hat{\pi}(a_1|s)\right) - g(\pi(a_1|s)) \\ &\geq 0 \\ &\geq g\left(\hat{\pi}(a_2|s)\right) - g(\pi(a_2|s)) \\ &= h\left(s, Q^\pi(s,a_2)\right). \end{aligned} \tag{7}$$

Let $h\left(s, Q^\pi(s,a)\right) = Q^\pi(s,a)$. We can get $Q^\pi(s,a_1) \geq Q^\pi(s,a_2)$ which means we can always find $q(s) \in \mathbb{R}$ such that

$Q^\pi(s,a_1) \geq q(s) \geq Q^\pi(s,a_2), \forall a_1 \in \mathcal{A}_1, \forall a_2 \in \mathcal{A}_2$. Thus,

$$\begin{aligned}
&\sum_{a\in\mathcal{A}} \hat{\pi}(a|s)Q^\pi(s,a) - \sum_{a\in A} \pi(a|s)Q^\pi(s,a) \\
&= \sum_{a_1\in\mathcal{A}_1} \left(\hat{\pi}\left(a_1|s\right) - \pi(a_1|s)\right) Q^\pi\left(s,a_1\right) + \sum_{a_2\in\mathcal{A}_2} \left(\hat{\pi}\left(a_2|s\right) - \pi(a_2|s)\right) Q^\pi(s,a_2) \\
&\geq \sum_{a_1\in\mathcal{A}_1} \left(\hat{\pi}\left(a_1|s\right) - \pi\left(a_1|s\right)\right) q(s) + \sum_{a_2\in\mathcal{A}_2} \left(\hat{\pi}\left(a_2|s\right) - \pi\left(a_2|s\right)\right) q(s) \\
&= q(s) \sum_{a\in\mathcal{A}} \left(\pi'(a|s) - \pi(a|s)\right) \\
&= 0.
\end{aligned} \tag{8}$$

Let $V_0(s) = V^\pi(s)$. And we denote the value function of following $\hat{\pi}$ for l steps then following π as $V_l(s) = E_{a\sim\hat{\pi}(.|s)}\left[E_{s',r|s,a}\left(r+\gamma V_{l-1}\left(s'\right)\right)\right]$ if $l \geq 1$.

First, we observe that

$$\begin{aligned}
V_1(s) &= E_{a\sim\hat{\pi}(.|s)}\left[E_{s',r|s,a}\left(r+\gamma V^\pi(s')\right)\right] \\
&= \sum_{a\in\mathcal{A}} \hat{\pi}(a|s)Q^\pi(s,a) \\
&\geqslant \sum_{a\in\mathcal{A}} \pi(a|s)Q^\pi(s,a) \\
&= V_0(s).
\end{aligned} \tag{9}$$

By induction, we assume $V_l(s) \geq V_{l-1}(s)$. Given that

$$\begin{aligned}
V_{l+1}(s) &= E_{a\sim\hat{\pi}}\left[E_{s',r|s,a}\left(r+V_l\left(s'\right)\right)\right], \\
V_l(s) &= E_{a\sim\hat{\pi}}\left[E_{s',r|s,a}\left(r+V_{l-1}\left(s'\right)\right)\right],
\end{aligned}$$

we have $V_{l+1}(s) \geq V_l(s)$.

Therefore, we can conclude that $V_{l+1}(s) \geq V_l(s), \forall l \geq 0$. We have $V_\infty(s) \geq V_0(s)$ which is $V^{\hat{\pi}}(s) \geq V^\pi(s)$. □

Based on the Proposition B.2, if we choose g as a logarithmic function and $h = \log(w(s,a)) + \log(d_\rho^\pi(s)) - \log(d_\rho^{\hat{\pi}}(s))$, we can easily verify that our MaskNet-based sampling approach is equivalent to sampling from a better policy $\hat{\pi}$.

□

B.3. Proof of Theorem 3.6

Proof. Given the fact that the refined policy π' is converged, (*i.e.,* the local one-step improvement is small $\mathbb{E}_{s\sim d_\mu^{\pi'}}\left[\max_a A^{\pi'}(s,a)\right] < \epsilon$), we have

$$\begin{aligned}
\epsilon &> \sum_{s\in\mathcal{S}} d_\mu^{\pi'}(s)\left[\max_a A^{\pi'}(s,a)\right] \\
&\geq \min_s \left(\frac{d_\mu^{\pi'}(s)}{d_\rho^{\pi^*}(s)}\right) \sum_s d_\rho^{\pi^*}(s) \max_a A^{\pi'}(s,a) \\
&\geq \left\|\frac{d_\rho^{\pi^*}}{d_\mu^{\pi'}}\right\|_\infty^{-1} \sum_{s,a} d_\rho^{\pi^*}(s)\pi^*(a|s)A^{\pi'}(s,a).
\end{aligned} \tag{10}$$

Based on the Performance Difference Lemma (Kakade & Langford, 2002), for two policies π^*, π' and a state distribution ρ, the performance difference is bounded by

$$V^{\pi^*}(\rho) - V^{\pi'}(\rho) = \frac{1}{1-\gamma} E_{s\sim d_\rho^{\pi^*}} E_{a\sim\pi^*(.|s)}\left[A^{\pi'}(s,a)\right]. \tag{11}$$

Then we have

$$\epsilon > (1-\gamma) \left\| \frac{d_\rho^{\pi^*}}{d_\mu^{\pi'}} \right\|_\infty^{-1} \left(V^\pi(\rho) - V^{\pi'}(\rho) \right). \tag{12}$$

Therefore, we have

$$V^{\pi^*}(\rho) - V^{\pi'}(\rho) \leq \frac{\epsilon}{1-\gamma} \left\| \frac{d_\rho^{\pi^*}}{d_\mu^{\pi'}} \right\|_\infty. \tag{13}$$

Due to $d_\mu^{\pi'}(s) \geq (1-\gamma)\mu(s)$, we further obtain

$$V^{\pi^*}(\rho) - V^{\pi'}(\rho) \leq \frac{\epsilon}{(1-\gamma)^2} \left\| \frac{d_\rho^{\pi^*}}{\mu} \right\|_\infty. \tag{14}$$

Since $\mu(s) = \beta d_\rho^{\hat{\pi}}(s) + (1-\beta)\rho(s) \geq \beta d_\rho^{\hat{\pi}}(s)$, we have

$$V^{\pi^*}(\rho) - V^{\pi'}(\rho) \leq \frac{\epsilon}{(1-\gamma)^2} \left\| \frac{d_\rho^{\pi^*}}{\beta d_\rho^{\hat{\pi}}} \right\|_\infty. \tag{15}$$

In our case, β is a constant (*i.e.,* a hyper-parameter), thus we could derive that

$$V^{\pi^*}(\rho) - V^{\pi'}(\rho) \leq \mathcal{O}\left(\frac{\epsilon}{(1-\gamma)^2} \left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\hat{\pi}}} \right\|_\infty\right), \tag{16}$$

which completes the proof. □

B.4. Analysis of Claim 1

Recall that Lemma 3.5 indicates that our MaskNet-based sampling approach is equivalent to sampling states from a better policy $\hat{\pi}$ compared with a random explanation sampling from the policy π, *i.e.,* $\eta(\hat{\pi}) \geq \eta(\pi)$. Let us denote the new initial distribution using our MaskNet-based sampling approach as μ. By Assumption 3.4, we have $\left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\hat{\pi}}} \right\|_\infty \leq \left\| \frac{d_\rho^{\pi^*}}{d_\rho^{\pi}} \right\|_\infty$. Using our explanation method introduces a smaller distribution mismatch coefficient than using a random explanation method. Therefore, we claim that using our explanation method, we are able to form a better initial distribution μ and tighten the upper bound in Theorem 3.6, *i.e.,* enhancing the agent's performance after refining.

C. Details of Evaluation

C.1. Implementation Details

Implementation of Our Method. We implement the proposed method using PyTorch (Paszke et al., 2019). We implement our method in four selected MuJoCo games based on Stable-Baselines3 (Raffin et al., 2021). We train the agents on a server with 8 NVIDIA A100 GPUs for all the learning algorithms. For all our experiments, if not otherwise stated, we use a set of default hyper-parameters for p, λ, and α (listed in Appendix C.3).

We implement the environment reset function similar to Ecoffet et al. (2019) to restore the environment to selected critical states. This method is feasible in our case, as we operate within simulator-based environments. However, in the real world, it may not be always possible to return to a certain state with the same sequences of actions due to the stochastic nature of state transition. It's important to note that our framework is designed to be versatile and is indeed compatible with a goal/state-conditioned policy approach such as Ecoffet et al. (2021). Given a trajectory with an identified most important state, we can select the most important state as the final goal and select the en-route intermediate states as sub-goals. Then we can train an agent to reach the final goal by augmenting each state with the next goal and giving a goal-conditioned reward once the next goal is reached until all goals are achieved.

Implementation of Baseline Methods. Regarding baseline approaches, we use the code released by the authors or implement our own version if the authors don't release the code. Specifically, as for StateMask, we use their released open-sourced code from `https://github.com/nuwuxian/RL-state_mask`. Regarding Jump-Start Reinforcement Learning, we use the implementation from `https://github.com/steventango/jumpstart-rl`.

C.2. Extra Introduction to Applications

Hopper. Hopper game is a captivating two-dimensional challenge featuring a one-legged figure comprising a torso, thigh, leg, and a single supporting foot (Erez et al., 2012). The objective is to propel the Hopper forward through strategic hops by applying torques to the three hinges connecting its body parts. Observations include positional values followed by velocities of each body part, and the action space involves applying torques within a three-dimensional action space. Under the `dense` reward setting, the reward system combines healthy reward, forward reward, and control cost. Under the `sparse` reward setting (Mazoure et al., 2019), the reward informs the x position of the hopper only if $x > 0.6$ in our experiments. The episode concludes if the Hopper becomes unhealthy. We use "Hopper-v3" in our experiments.

Walker2d. Walker2d is a dynamic two-dimensional challenge featuring a two-legged figure with a torso, thighs, legs, and feet. The goal is to coordinate both sets of lower limbs to move in the forward direction by applying torques to the six hinges connecting these body parts. The action space involves six dimensions, allowing exert torques at the hinge joints for precise control. Observations encompass positional values and velocities of body parts, with the former preceding the latter. Under the `dense` reward setting, the reward system combines a healthy reward bonus, forward reward, and control cost. Under the `sparse` reward setting (Mazoure et al., 2019), the reward informs the x position of the hopper only if $x > 0.6$ in our experiments. The episode concludes if the walker is deemed unhealthy. We use "Walker2d-v3" in our experiments and normalize the observation when training the DRL agent.

Reacher. Reacher is an engaging two-jointed robot arm game where the objective is to skillfully maneuver the robot's end effector, known as the fingertip, towards a randomly spawned target. The action space involves applying torques at the hinge joints. Observations include the cosine and sine of the angles of the two arms, the coordinates of the target, angular velocities of the arms, and the vector between the target and the fingertip. It is worth noting that there is no sparse reward implementation of Reacher-v2 in Mazoure et al. (2019). The reward system comprises two components: "reward distance" indicating the proximity of the fingertip to the target, and "reward control" penalizing excessive actions with a negative squared Euclidean norm. The total reward is the sum of these components, and an episode concludes either after 50 timesteps with a new random target or if any state space value becomes non-finite. We use "Reacher-v2" in our experiments.

HalfCheetah. HalfCheetah is an exhilarating 2D robot game where players control a 9-link cheetah with 8 joints, aiming to propel it forward with applied torques for maximum speed. The action space contains six dimensions, that enable strategic movement. Observations include positional values and velocities of body parts. Under the `dense` reward setting, the reward system balances positive "forward reward" for forward motion with "control cost" penalties for excessive actions. Under the `sparse` reward setting (Mazoure et al., 2019), the reward informs the x position of the hopper only if $x > 5$ in our experiments. Episodes conclude after 1000 timesteps, offering a finite yet thrilling gameplay experience. We use "HalfCheetah-v3" in our experiments and normalize the observation when training the DRL agent.

Selfish Mining. Selfish mining is a security vulnerability in blockchain protocols, identified by Eyal & Sirer (2018). When a miner holds a certain amount of computing power, they can withhold their freshly minted blocks from the public blockchain, thereby initiating a fork that is subsequently mined ahead of the official public blockchain. With this advantage, the miner can introduce this fork into the network, overwriting the original blockchain and obtaining more revenue.

To find the optimal selfish mining strategies, Bar-Zur et al. (2023) proposed a deep reinforcement learning model to generate a mining policy. The policy takes the current chain state as the input and chooses from the three pre-determined actions, *i.e.,* adopting, revealing, and mining. With this policy network, the miner can obtain more mining rewards compared to using heuristics-based strategies.

We train a PPO agent in the blockchain model developed by Bar-Zur *et al.* (git, d). The network architecture of the PPO agent is a 4-layer Multi-Layer Perceptron (MLP) with a hidden size of 128, 128, 128, and 128 in each layer. We adopt a similar network structure for training our mask network. The whale transaction has a fee of 10 with the occurring probability of 0.01 while other normal transactions have a fee of 1. The agent will receive a positive reward if his block is accepted and will be penalized if his action is determined to be unsuccessful, *e.g.,* revealing a private chain.

In our selfish mining task (Bar-Zur et al., 2023), three distinct actions are defined as follows:

Adopt l: The miner chooses to adopt the first l blocks in the public chain while disregarding their private chain. Following this, the miner will continue their mining efforts, commencing from the last adopted block.

Reveal l: This action becomes legal when the miner's private chain attains a length of at least l. The consequence of this

action may result in either the creation of an active fork in the public chain or the overriding of the public chain.

Mine: This action simply involves continuing with the mining process. Once executed, a new block is mined and subsequently added to either the private chain of the rational miner or to the public chain, contingent on which entity successfully mined the block.

CAGE Challenge 2. To inspire new methods for automating cyber defense, the Technical Cooperation Program (TTCP) launched the Autonomous Cyber Defence Challenge (CAGE Challenge) to produce AI-based blue teams for instantaneous response against cyber attacks (CAGE, 2022). The goal is to create a DRL blue agent to protect a network against a red agent. The action space of the blue agent includes monitoring, analyzing, decoyApache, decoyFemitter, decoyHarakaSMPT, decoySmss, decoySSHD, decoySvchost, decoyTomcat, removing, and restoring. Note that the blue agent can receive a *negative reward* when the red agent gets admin access to the system (and continues to receive negative rewards as the red agent maintains the admin access). We use CAGE challenge 2 for our evaluation.

We choose the champion scheme proposed by Cardiff University (git, c) in CAGE challenge 2 (git, b). The target agent is a PPO-based blue agent to defend a network against the red agent "B-line". The trail has three different lengths, *i.e.,* 30, 50, and 100. The final reward is the sum of the average rewards of these three different lengths.

The action set of the blue agent is defined as follows.

Monitor: The blue agent automatically collects the information about flagged malicious activity on the system and reports network connections and associated processes that are identified as malicious.

Analyze: The blue agent analyzes the information on files associated with recent alerts including signature and entropy.

DecoyApache, DecoyFemitter, DecoyHarakaSMPT, DecoySmss, DecoySSHD, DecoySvchost, DecoyTomcat: The blue agent sets up the corresponding decoy service on a specified host. An alert will be raised if the red agent accesses the decoy service.

Remove: The blue agent attempts to remove red from a host by destroying malicious processes, files, and services.

Restore: The blue agent restores a system to a known good state. Since it significantly impacts the system's availability, a reward penalty of -1 will be added when executing this action.

Autonomous Driving. Deep reinforcement learning has been applied in autonomous driving to enhance driving safety. One representative driving simulator is MetaDrive (Li et al., 2022). A DRL agent is trained to guide a vehicle safely and efficiently to travel to its destination. MetaDrive converts the Birds Eye View (BEV) of the road conditions and the sensor information such as the vehicle's steering, direction, velocity, and relative distance to traffic lanes into a vector representation of the current state. The policy network takes this state vector as input and yields driving actions, including accelerating, braking, and steering commands. MetaDrive employs a set of reward functions to shape the learning process. For instance, penalties are assigned when the agent collides with other vehicles or drives out of the road boundary. To promote smooth and efficient driving, MetaDrive also incorporates rewards to encourage forward motion and the maintenance of an appropriate speed.

We select the "Macro-v1" environment powered by the MetaDrive simulator (Li et al., 2022). The goal of the agent is to learn a deep policy to successfully cross the car flow and reach the destination. We train the target agent and our mask network by the PPO algorithm following the implementation of DI-drive (drive Contributors, 2021). The environment receives normalized action to control the target agent $\mathbf{a} = [a_1, a_2] \in [-1, 1]^2$. The action vector $\mathbf{a}$ will then be converted to the steering (degree), acceleration (hp), and brake signal (hp).

Malware Mutation. DRL has been used to assess the robustness of ML-based malware detectors. For example, Anderson et al. (2018) propose a DRL-based approach to attack malware detectors for portable executable (PE) files. We use the "Malconv" gym environment Raff et al. (2017) implemented in (git, a) for our experiments. We train a DRL agent based on Tianshou framework (Weng et al., 2022). The input of the DRL agent is a feature vector of the target malware and outputs the corresponding action to guide the malware manipulation. We present the action set of the MalConv gym environment in Table 2 for ease of comprehension in the case study section. A big reward of 10 is provided when evading detection.

The reward mechanism of the "Malconv" environment is as follows. Initially, the malware detection model will provide a score sc_0 of the current malware. If sc_0 is lower than some threshold, the malware has already evaded the detection. Otherwise, the DRL agent will take some mutation actions to bypass the detection. At step t, after executing the agent's

Table 2. Action set of the MalConv gym environment.

Action index	Action meaning
0	"modify_machine_type"
1	"pad_overlay"
2	"append_benign_data_overlay"
3	"append_benign_binary_overlay"
4	"add_bytes_to_section_cave"
5	"add_section_strings"
6	"add_section_benign_data"
7	"add_strings_to_overlay"
8	"add_imports"
9	"rename_section"
10	"remove_debug"
11	"modify_optional_header"
12	"modify_timestamp"
13	"break_optional_header_checksum"
14	"upx_unpack"
15	"upx_pack"

Table 3. Hyper-parameter choices in Experiment I-V. "Selfish" represents Selfish Mining. "Cage" represents Cage Challenge 2. "Auto" represents Autonomous Driving. "Malware" represents Malware Mutation.

Hyper-parameter	Hopper	Walker2d	Reacher	HalfCheetah	Selfish	Cage	Auto	Malware
p	0.25	0.25	0.50	0.50	0.25	0.50	0.25	0.50
λ	0.001	0.01	0.001	0.01	0.001	0.01	0.01	0.01
α	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001

action, the malware detection model will provide a new score sc_t. If sc_t is lower than some threshold, the mutation is successful and a big reward of 10 will be given. Otherwise, the reward will be $sc_0 - sc_t$. The maximum allowed number of steps is 10.

C.3. Additional Experiment Results

Hyper-parameter Choices in Experiment I-V. Table 3 summarizes our hyper-parameter choices in Experiment I-V. For all applications, we choose the coefficient of the intrinsic reward for training the mask network α as 0.01. The hyper-parameters p and λ for our refining method vary by application.

Fidelity Scores in Experiment I. Figure 5 shows the fidelity score comparison across all explanation methods. We have three key observations. First, We observe that our explanation method has similar fidelity scores with StateMask across all applications, empirically indicating the equivalence of our explanation method with StateMask. Second, we observe that our explanation method and StateMask have higher fidelity scores than random explanation across all applications, indicating that the mask network provides more faithful explanations for the target agents.

Efficiency Comparison in Experiment II. Table 4 reports the efficiency evaluation results when training a mask network using StateMask and our method. We observe that it takes 16.8% less time on average to train a mask network using our method than using StateMask, which shows the advantage of our method with respect to efficiency.

Comparison with Self-Imitation Learning. We compare `RICE` against the self-imitation learning (SIL) approach (Oh et al., 2018) across four MuJoCo games. We present the results presented in Table 5. These experiment results demonstrate that `RICE` consistently outperforms the self-imitation learning method. While self-imitation learning has the advantage of encouraging the agent to imitate past successful experiences by prioritizing them in the replay buffer, it cannot address scenarios where the agent (and its past experience) has errors or sub-optimal actions. In contrast, `RICE` constructs a mixed initial distribution based on the identified critical states (using explanation methods) and encourages the agent to explore the new initial states. This helps the agent escape from local minima and break through the training bottlenecks.

Impact of Other Explanation Methods. We investigate the impact of other explanation methods (*i.e.,* AIRS (Yu et al., 2023) and Integrated Gradients (Sundararajan et al., 2017)) on four Mujoco games. we fix the refining method and use

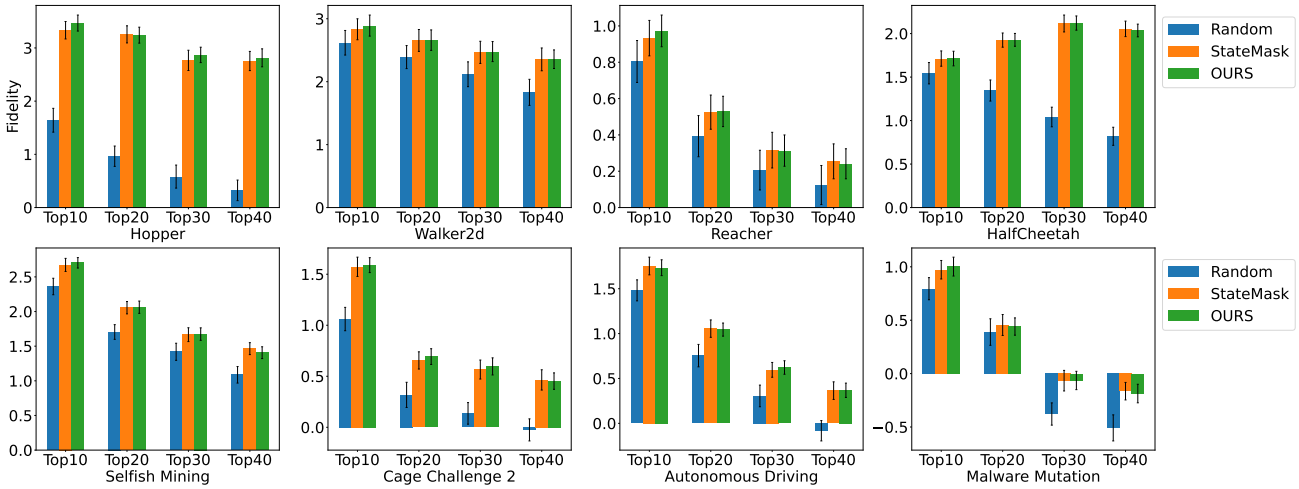


Figure 5. Fidelity scores for explanation generated by baseline methods and our proposed explanation method. Note that a higher score implies higher fidelity.

Table 4. Efficiency comparison when training the mask network. We report the number of seconds when training the mask using a fixed number of samples. "Selfish" represents Selfish Mining. "Cage" represents Cage Challenge 2. "Auto" represents Autonomous Driving. "Malware" represents Malware Mutation.

Applications	Hopper	Walker2d	Reacher	HalfCheetah	Selfish	Cage	Auto	Malware
Num. of samples	3×10^5	3×10^5	3×10^5	3×10^5	1.5×10^6	1×10^7	2443260	32349
StateMask	15393	2240	8571	1579	9520	79382	109802	50775
Ours	**12426**	**1899**	**7033**	**1317**	**8360**	**65400**	**88761**	**41340**

different explanation methods to identify critical steps for refinement. The results are reported in Table 6. We observe that using the explanation generated by our mask network, the refining achieves the best outcome across all four applications. Using other explanation methods (Integrated Gradients and AIRS), our framework still achieves better results than the random baseline, suggesting that our framework can work with different explanation method choices.

Sensitivty of p and λ in Hopper game with an imitated PPO agent. We report the sensitivity of hyper-parameters p and λ in Hopper game with an imitated PPO agent in Figure 6. We observe that in general, a mixture probability of $p = 0.25$ or $p = 0.5$ is a better choice. An RND bonus can facilitate the agent with faster refinement.

Sensitivity of Hyper-parameters p and λ. We provide the sensitivity results of p in all applications in Figure 7. We observe that generally a mixture probability of $p = 0.25$ or $p = 0.5$ is a good choice. Additionally, recall that we need to use the hyper-parameter λ to balance the scale of the "true" environment reward and the exploration bonus. We test the sensitivity of λ from the space $\{0.1, 0.01, 0.001\}$. Figure 8 reports the agent's performance after refining under different settings of λ. We observe that our retaining method is insensitive to the choice of λ. The agent's performance does not vary a lot with different settings of λ. But $\lambda = 0.01$ gives the best performance in all applications except selfish mining.

Sensitivity of α. Recall that under certain assumptions, we are able to simplify the design of StateMask. We propose an intrinsic reward mechanism to encourage the mask network to blind more states without sacrificing performance. The hyper-parameter α is then introduced to balance the performance of the perturbed agent and the need for encouraging blinding. We test the sensitivity of α from the space $\{0.01, 0.001, 0.0001\}$ and report the fidelity scores under different settings of α in Figure 9. We observe that though the value of α varies, the fidelity score does not change much.

C.4. Evaluation Results of MuJoCo Games with Sparse Rewards

Results of SparseWalker2d. First, we compare our refining method with other baseline methods (*i.e.,* PPO fine-tuning, StateMask-R, and JSRL) in the SparseWalker2d game. Figure 10 shows that our refining method is able to help the DRL agent break through the bottleneck with the highest efficiency compared with other baseline refining methods. Additionally, by replacing our explanation method with a random explanation, we observe that the refining performance is getting worse.

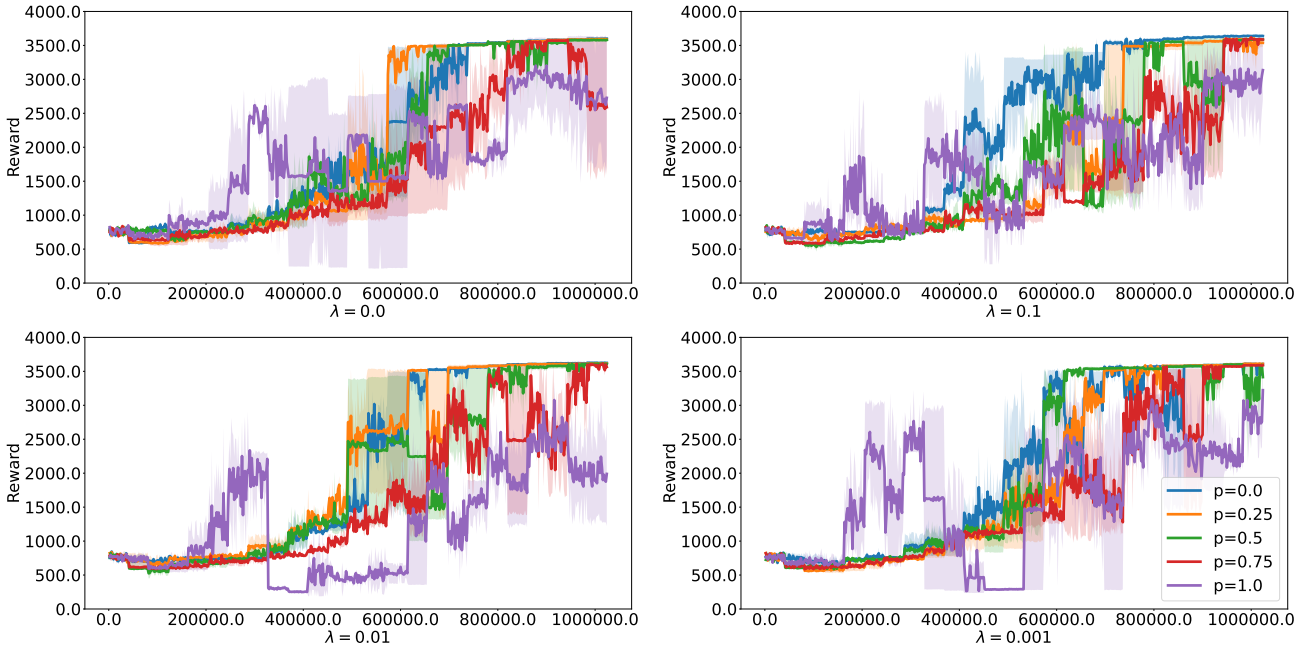


Figure 6. Sensitivity results of hyper-parameters p and λ in Hopper game with an imitated PPO agent. We vary the hyper-parameter λ from $\{0, 0.1, 0.01, 0.001\}$ and record the performance of the agent after refining. A smaller choice of λ means a smaller reward bonus for exploration.

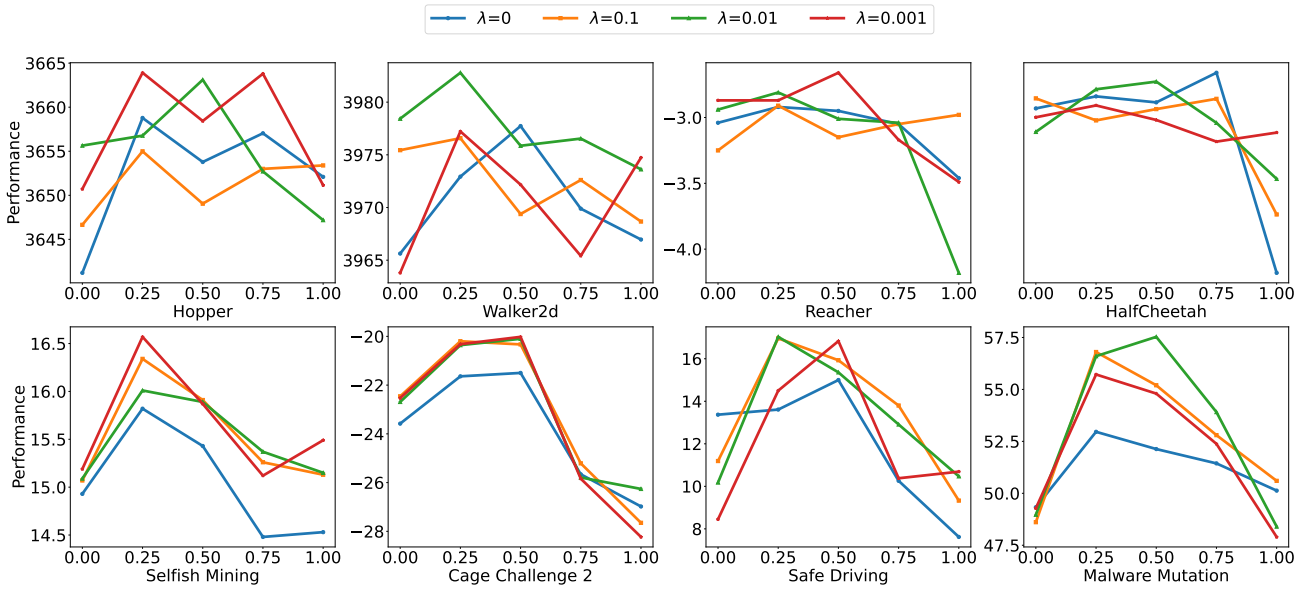


Figure 7. Sensitivity results of hyper-parameter p in all applications. We vary the hyper-parameter p from $\{0, 0.25, 0.5, 0.75, 1\}$ under different λ, and record the performance of the agent after refining. When $p = 0$, refining starts from the default initial states of the environment. When $p = 1$, refining starts exclusively from critical states. We show that the "mixed" initial state distribution helps to achieve a better performance.

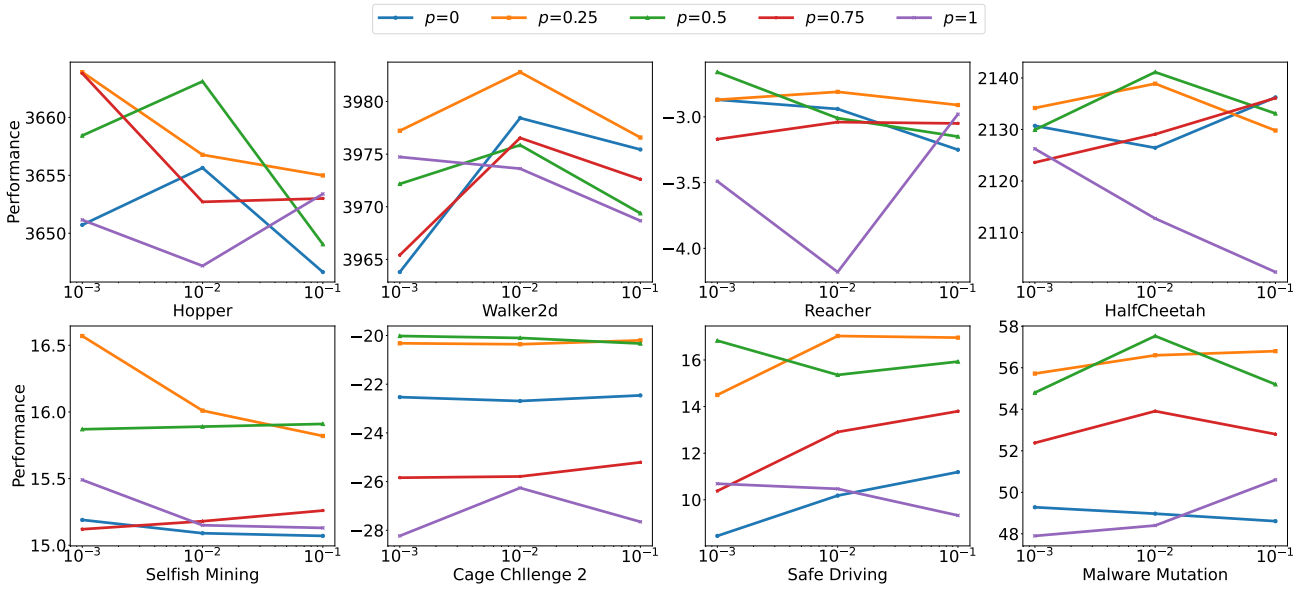


Figure 8. Sensitivity results of hyper-parameter λ. We vary the hyper-parameter λ from $\{0.1, 0.01, 0.001\}$ and record the performance of the agent after refining. A smaller choice of λ means a smaller reward bonus for exploration.

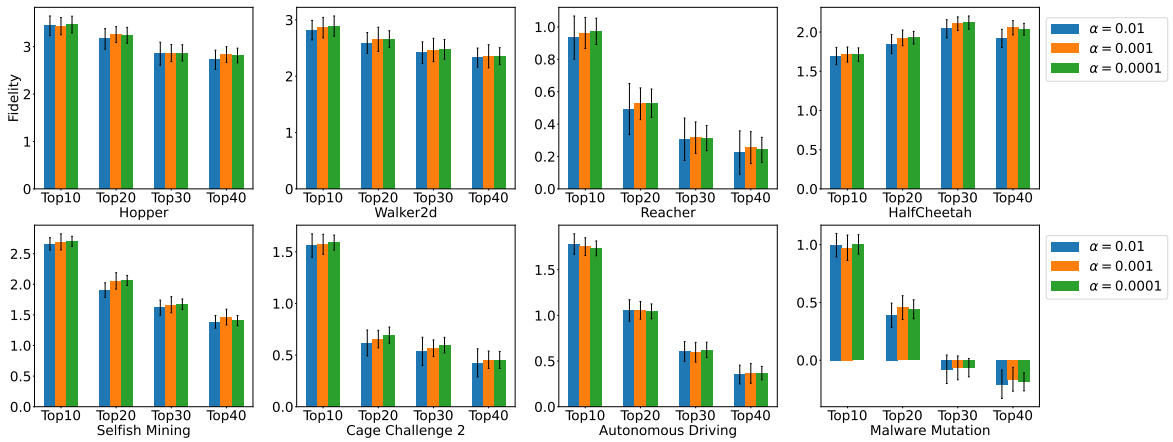


Figure 9. Sensitivity results of hyper-parameter α. We vary the hyper-parameter α from $\{0.01, 0.001, 0.0001\}$ and record the fidelity scores of the mask network trained under different settings of α. A higher fidelity score means a higher fidelity.

Table 5. Performance comparison between Self-Imitation Learning (SIL) and RICE on four MuJoCo tasks.

Method	Hopper	Walker2d	Reacher	HalfCheetah
SIL	3646.46 (23.12)	3967.66 (1.53)	-2.87 (0.09)	2069.80 (3.44)
Ours	**3663.91 (20.98)**	**3982.79 (3.15)**	**-2.66 (0.03)**	**2138.89 (3.22)**

Table 6. Performance comparison when using different explanation methods across four MuJoCo tasks.

Task	Random Explanation	Integrated Gradients	AIRS	Ours
Hopper	3648.98 (39.06)	3653.24 (14.23)	3654.49 (8.12)	**3663.91 (20.98)**
Walker2d	3969.64 (6.38)	3972.15 (4.77)	3976.35 (2.40)	**3982.79 (3.15)**
Reacher	-3.11 (0.42)	-2.99 (0.31)	-2.89 (0.19)	**-2.66 (0.03)**
HalfCheetah	2132.01 (0.76)	2132.81 (0.83)	2133.98 (2.52)	**2138.89 (3.22)**

Sensitivity of p and λ. We report the sensitivity of hyper-parameters p and λ in the three MuJoCo games with sparse rewards in Figure 11, Figure 12, and Figure 13. We have the following observations: First, generally, a mixed probability p within the range of 0.25 and 0.5 would be a good choice. Second, the refining benefits from the exploration bonus in the sparse MuJoCo games. Third, PPO fine-tuning cannot guarantee that the refined agent can achieve a good performance. Especially in SparseWalker2d game, we observe that ppo fine-tuning cannot break through the training bottleneck of the DRL agent.

C.5. Qualitative Analysis

We do a qualitative analysis of the autonomous driving case to understand how RICE impacts agent behavior and performance. We visualize the agent's behavior before and after refining the agent. Figure 14(a) shows a trajectory wherein the target agent (depicted by the green car) fails to reach its destination due to a collision with a pink car on the road. Given the undesired outcome, we use our method to identify the critical steps that contribute to the final (undesired) outcome. The important steps are highlighted in red color. Our method identifies the important step as the one when the green car switches across two lanes into the lane of the pink car. The critical state is reasonable because this early step allows the green car to switch lanes to avoid the collision. Based on the provided explanation, we apply our refining method to improve the target agent. The trajectory after refining is shown in Figure 14(b). It shows that after refining, the refined agent (the green car) successfully identifies an alternative path to reach the destination while avoiding collision.

D. Case Study: Malware Mutation

D.1. Design Intuitions

First, we use malware mutation as a case study to confirm our design intuitions before the proposed refining method. Recall that the refining method contains three important ideas. First, we integrate the explanation result (identified critical step) into the refining process. Second, we design a mixed initial state distribution to guide the refining of the target agent. Third, we encourage the agent to perform exploration for diverse states during the refining phase. In the following, we create multiple baselines by gradually adding these ideas to a naive baseline to show the contribution of each idea. We also provide evidence to support our stance against overfitting. Table 7 summarizes the results.

To start, the original agent is trained for 100 epochs until convergence. We test the target agent for 500 runs, resulting in an average evasion probability of 33.8%. To extract behavioral patterns, we perform a frequency analysis on the mutation actions taken by the agent across all 500 runs. As shown in the first row of Table 7, there is a clear preference for A_4 (*i.e.,* "add_bytes_to_section_cave"). A complete list of the possible actions (16 in total) is shown in Table 2 (Appendix).

Continue Learning w/o Explanation. The most common refining method is to lower the learning rate and continue training. We continue to train this target agent using the PPO algorithm for an additional 30 epochs and evaluate its performance over 500 runs. This yields an average evasion probability of 38.8% (second row in Table 7). It is worth noting that A_4 (*i.e.,* "add_bytes_to_section_cave") remains the most frequently selected action.

Leverage Explanation Results for Refining. Subsequently, we assess the refining outcome by incorporating our explanation result into the refining process. Specifically, we initiate the refining exclusively *from the critical steps* identified by the explanation method. For this setting, we do not perform exploration.

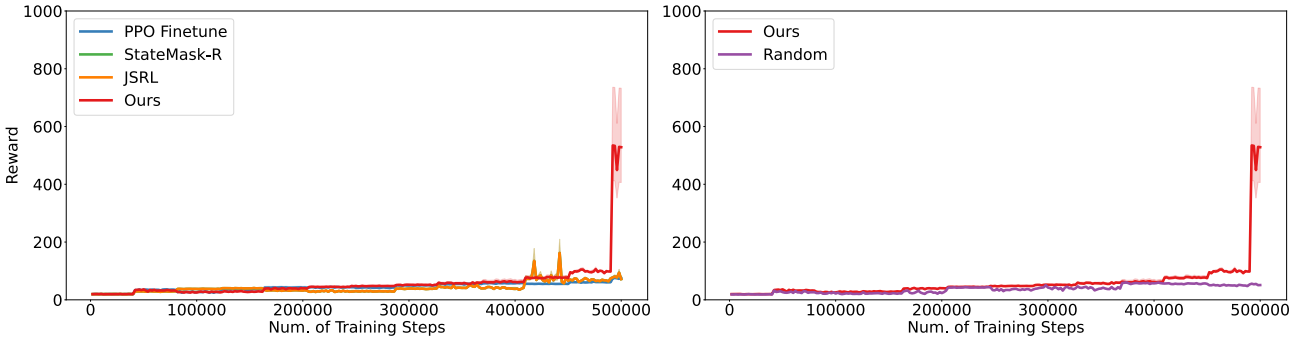


Figure 10. Agent Refining Performance in the SparseWalker2d Games. For the left figure, we fix the explanation method to our method (mask network) if needed while varying refining methods. For the right figure, we fix the refining method to our method while varying the explanation methods.

Table 7. **Malware Mutation Case Study**—We evaluate the evasion probability of the agent under different settings and count the corresponding action frequencies.

Refine Setting	Test Setting	Action Frequency	Evasion
Original agent w/o refinement	From default initial S	$\{A_4$: 4,914, A_9: 5$\}$	33.8%
Continue training	From default initial S	$\{A_4$: 2,590, A_7: 55, A_1: 99, A_9: 95$\}$	38.8%
Refine from critical states	From critical states	$\{A_{12}$: 2,546, A_5: 138, A_4: 32, A_9: 8$\}$	50.8%
	From default initial S	$\{A_{12}$: 4,728, A_5: 62$\}$	36.2%
Refine from mixed initial state dist.	From default initial S	$\{A_4$: 1,563, A_{12}: 1,135, A_5: 332, A_6: 12$\}$	58.4%
Refine from mixed initial state dist. + exploration	From default initial S	$\{A_5$: 2,448, A_7: 165, A_{12}: 138, A_4: 6$\}$	68.2%

During the test phase, we explore two testing settings. First, we artificially reset the test environment to start from these critical steps. We find that evasion probability surges to 50.8%. A_{12} (*i.e.,* "modify_timestamp") becomes as the most frequently chosen action. This indicates the refined agent learns a policy when encountering the critical state again. However, for more realistic testing, we need to set the test environment to the default initial state (*i.e.,* the correct testing condition). Under this setting, we find the evasion probability diminishes to 36.2%. This stark contrast in results shows evidence of overfitting. The refined agent excels at solving the problem when starting from critical steps but falters when encountering the task from default initial states.

Impact of Mixed Initial State Distribution. Given the above result, we further build a baseline by refining from the proposed mixed initial state distribution (*i.e.,* blending the default initial state distribution with the critical states). For this setting, we also do not perform exploration. Through 500 runs of testing, we observe a notable improvement, with the average evasion probability reaching 58.4% (from the previous baseline's 36.2%). Furthermore, the action frequency pattern has also undergone a shift. It combines the preferred actions from the two previous refining strategies, highlighting the frequent selection of both A_4 and A_{12}.

Impact of Exploration. Finally, we explore the impact of exploration. This baseline represents the complete version of our proposed system by adding the exploration step and using the mixed initial distribution. We notice that the average evasion probability across 500 runs has a major increase, reaching 68.2%. The most frequent action now is A_5 (*i.e.,* "add_section_strings"). A_4 and A_{12} are still among the top actions but their frequencies are lowered. This shows the benefits of exploring previously unseen states and diverse mutation paths. In return, the refined agent is able to get out of the local minima to identify more optimal policies.

D.2. Discovery of Design Flaws

Additionally, our explanation results have led to the discovery of design flaws in the malware mutation application (Raff et al., 2017). We will further explain how we use `RICE` to identify these problems.

Questions and Intuitions. When using `RICE` to explain the malware mutation process, we observe a scenario where the agent constantly chooses the same action "upx_pack" in multiple consecutive steps. According to the agent, these actions

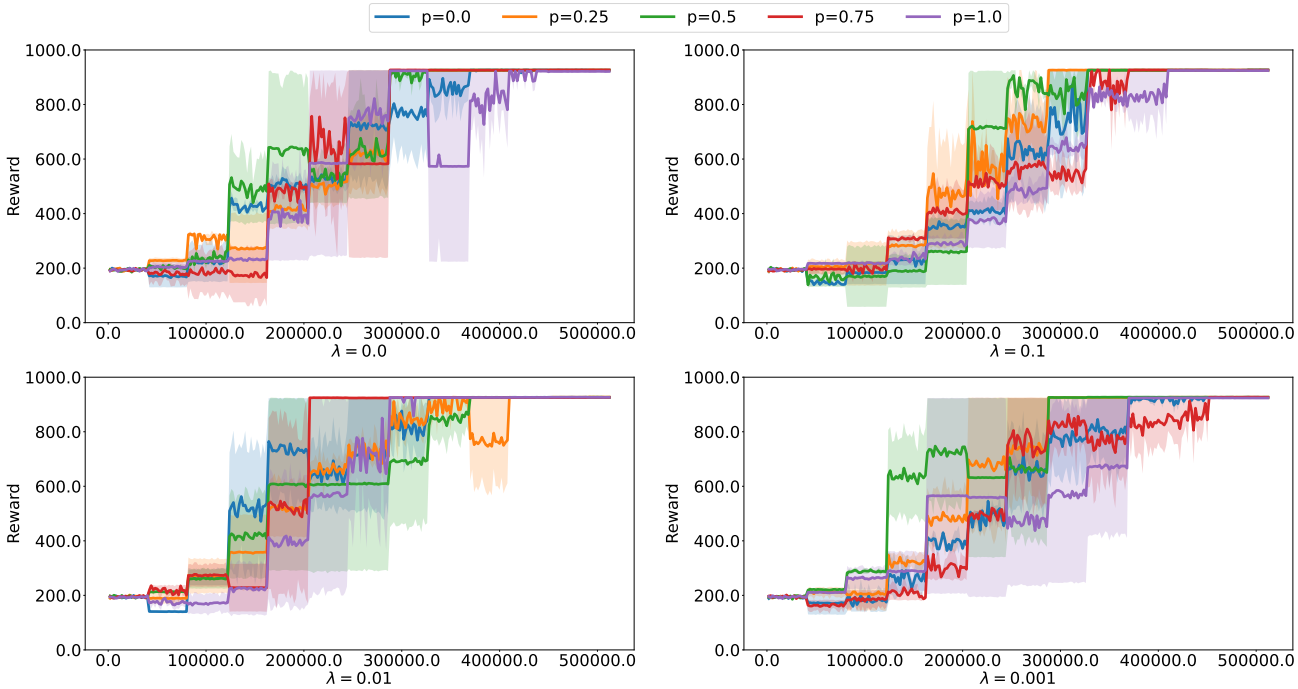


Figure 11. Sensitivity results of hyper-parameter λ in SparseHopper game. We vary the hyper-parameter λ from $\{0, 0.1, 0.01, 0.001\}$ and record the performance of the agent after refining. A smaller choice of λ means a smaller reward bonus for exploration.

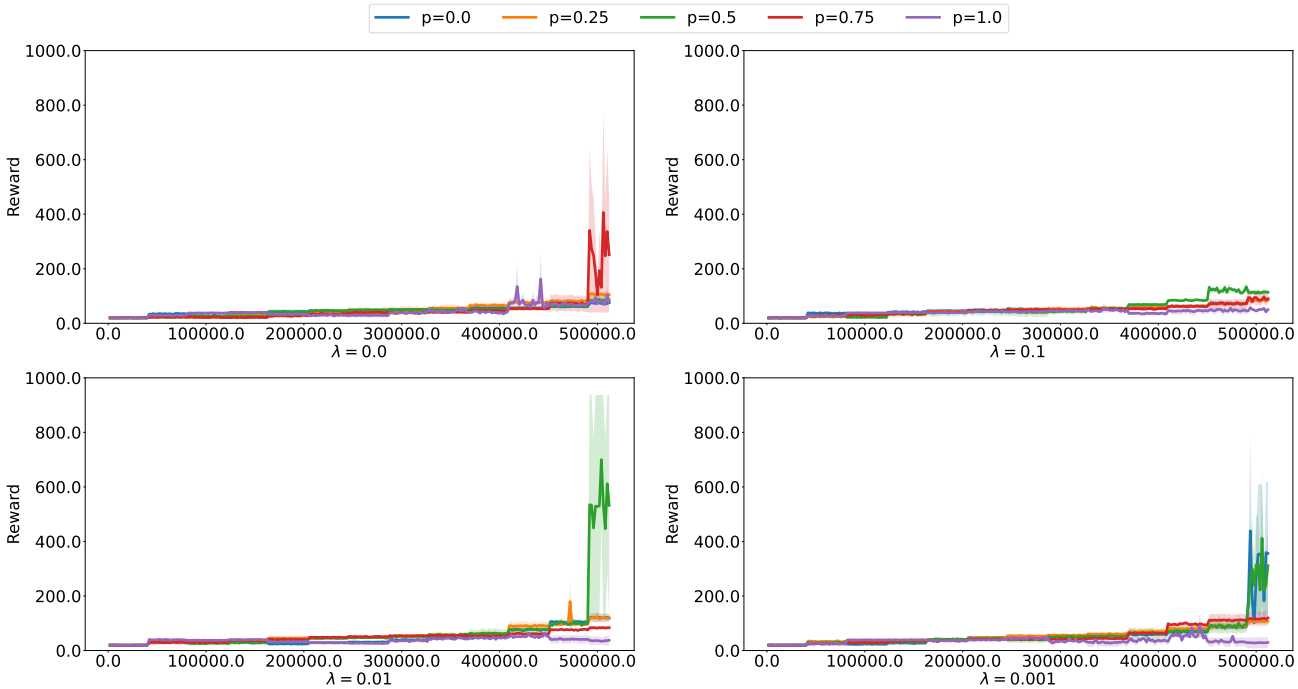


Figure 12. Sensitivity results of hyper-parameter λ in SparseWalker2d game. We vary the hyper-parameter λ from $\{0, 0.1, 0.01, 0.001\}$ and record the performance of the agent after refining. A smaller choice of λ means a smaller reward bonus for exploration.

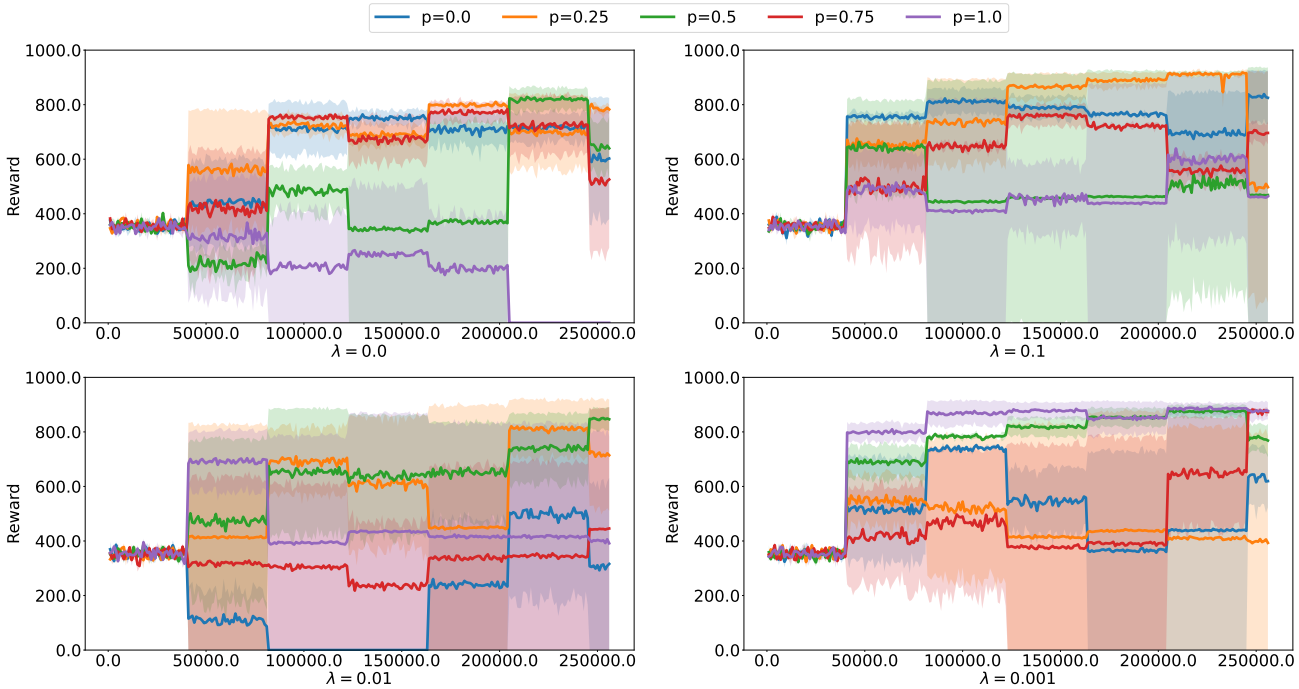


Figure 13. Sensitivity results of hyper-parameter λ in SparseHalfCheetah game. We vary the hyper-parameter λ from $\{0, 0.1, 0.01, 0.001\}$ and record the performance of the agent after refining. A smaller choice of λ means a smaller reward bonus for exploration.

receive a similar reward. However, RICE (our mask network) returns different "explanations" for these steps (*i.e.,* they have highly different importance scores). According to RICE, only the first action holds a high importance score, while the other consecutive actions barely have an impact on the final reward (*i.e.,* they appear redundant). This raises the question: why does the agent assign a similar reward to these consecutive steps in the first place?

Another interesting observation is from refining experiments. We find that PPO-based refining cannot yield substantial improvements. While we have expected that these methods do not perform as well as ours (given our exploration step), the difference is still bigger than we initially expected. This motivates us to further examine the reward function design to explore whether it has inadvertently discouraged the DRL agent from finding good evasion paths.

Problems of Reward Design. Driven by the intuitions above, we examined the reward design and identified two problems. Firstly, the reward mechanism is inherently *non-Markovian* which deviates from the expectation of a typical reinforcement learning (RL) framework. In typical RL settings, rewards are contingent on the current state s and the next state s'. However, the current design computes the reward based on the *initial state* s_0 and the subsequent state s'. Consequently, this may assign an identical reward for the same action (*e.g.,* "upx_pack") in consecutive steps. This non-Markovian nature of the reward mechanism can mislead the DRL agent and hurt its performance.

Second, we find that the *intermediate rewards* exhibit unusually high sparsity, *i.e.,* many intermediate rewards tend to have a value close to zero, which poses a significant challenge for the PPO algorithm to learn a good policy based on such intermediate rewards. Agents refined with these methods can be easily trapped in local minima.

Fixing the Problematic Reward Design. Based on these insights, we fix the bugs in the reward design with two simple steps: (1) We make the reward function Markovian, which depends only on the current state and the next state. (2) We perform scaling on the intermediate reward with a coefficient of 3. After that, we re-run an experiment to evaluate the correctness of our modifications. We train a DRL agent for 100 epochs with the same parameters under the new reward design and test its performance over 3 trials of 500 runs. The experiment shows that the evasion probability of the agent under the new reward design jumps from 42.2% (using the old reward function, see Table 1) to 72.0%, which further confirms our intuitions. This case study illustrates how developers can use RICE to debug their system and improve their designs.

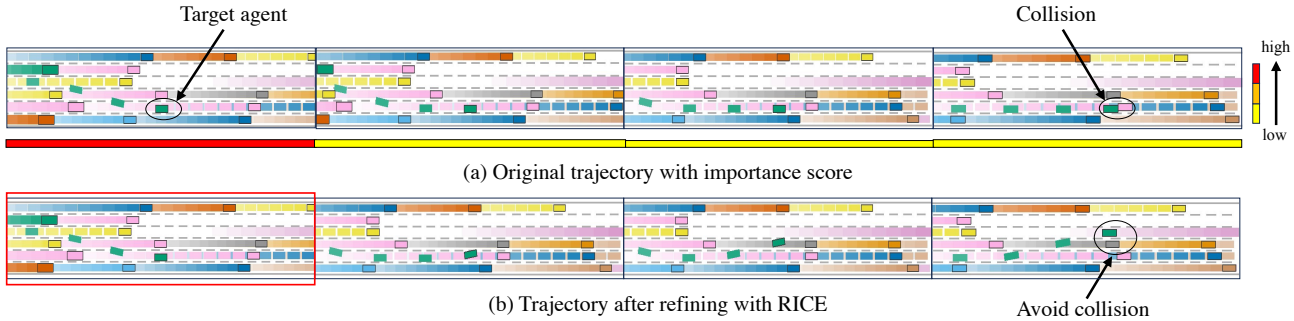


Figure 14. (a): In the original trajectory, the target agent (the green car) eventually collides with the pink car, which is an undesired outcome. Each time step is marked with a different color: "yellow" indicates the least important step and "red" represents the most important step. (b): We highlight the critical states identified by our explanation method and the corresponding outcome after refining. Using our explanation method, the target agent (the green car) successfully avoids collision.

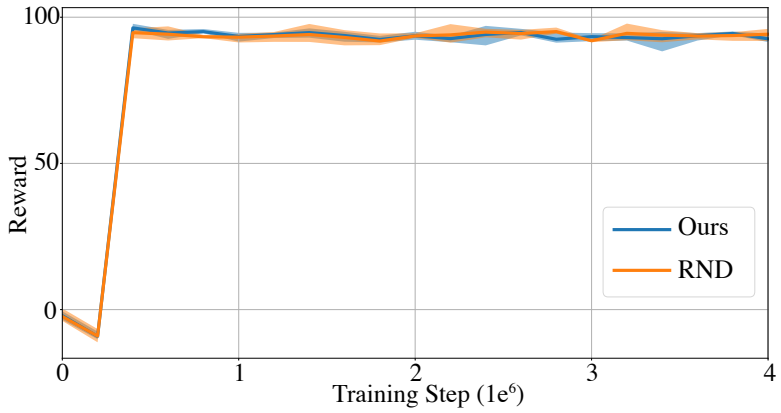


Figure 15. Refining performance with our method and RND method in MountainCarContinuous-v0 game. The state coverage of the pre-trained policy is limited to a small range around the initial point.

E. Limitation

We use the continuous "Mountain Car" environment (mou) as a negative control task to illustrate a scenario where `RICE` does not work well. In this "extreme" case, Assumption 3.2 is completely broken since the state coverage of the pre-trained agent is limited to a small range around the initial point. In this experiment, we train a target agent using Proximal Policy Optimization (PPO) for 1 million steps. The results show that the policy performance remained poor, with the agent frequently getting trapped at the starting point of the environment. In such cases where the original policy fails to learn an effective strategy, the role of explanations becomes highly limited. Since `RICE` relies on the identified critical states to enhance the policy, if the policy itself is extremely weak (*i.e.,* not satisfying Assumption 3.2), then the explanations will not be meaningful, which further huts the refinement. In the case of the Mountain Car experiment, `RICE` essentially reduces to being equivalent to Random Network Distillation (RND) due to the lack of meaningful explanation. We show the result when refining the pre-trained agent using our method and RND in Figure 15.